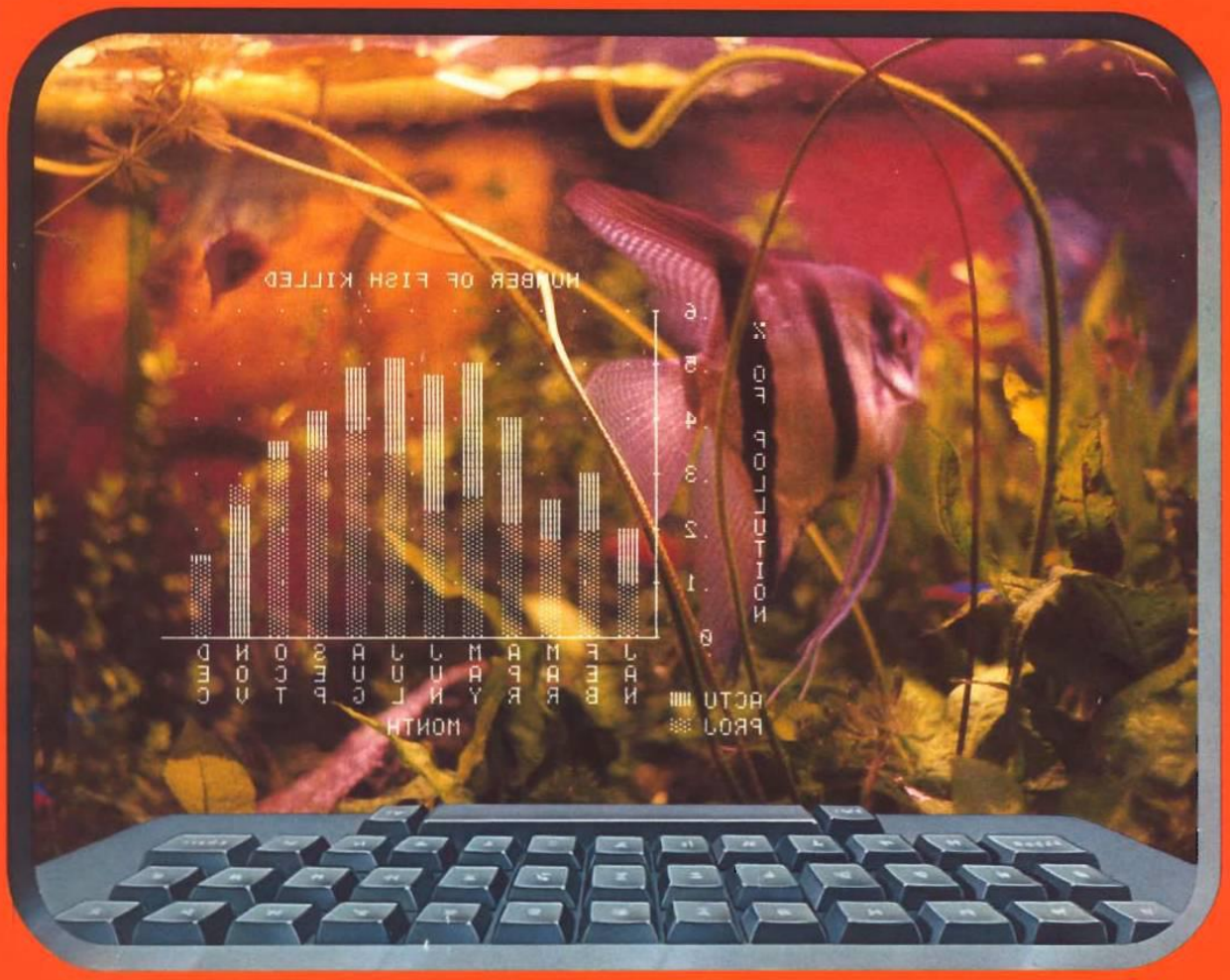
NO. 57 FEBRUARY 1983

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £2.00

MICRO™

Advancing Computer Knowledge



Language Feature

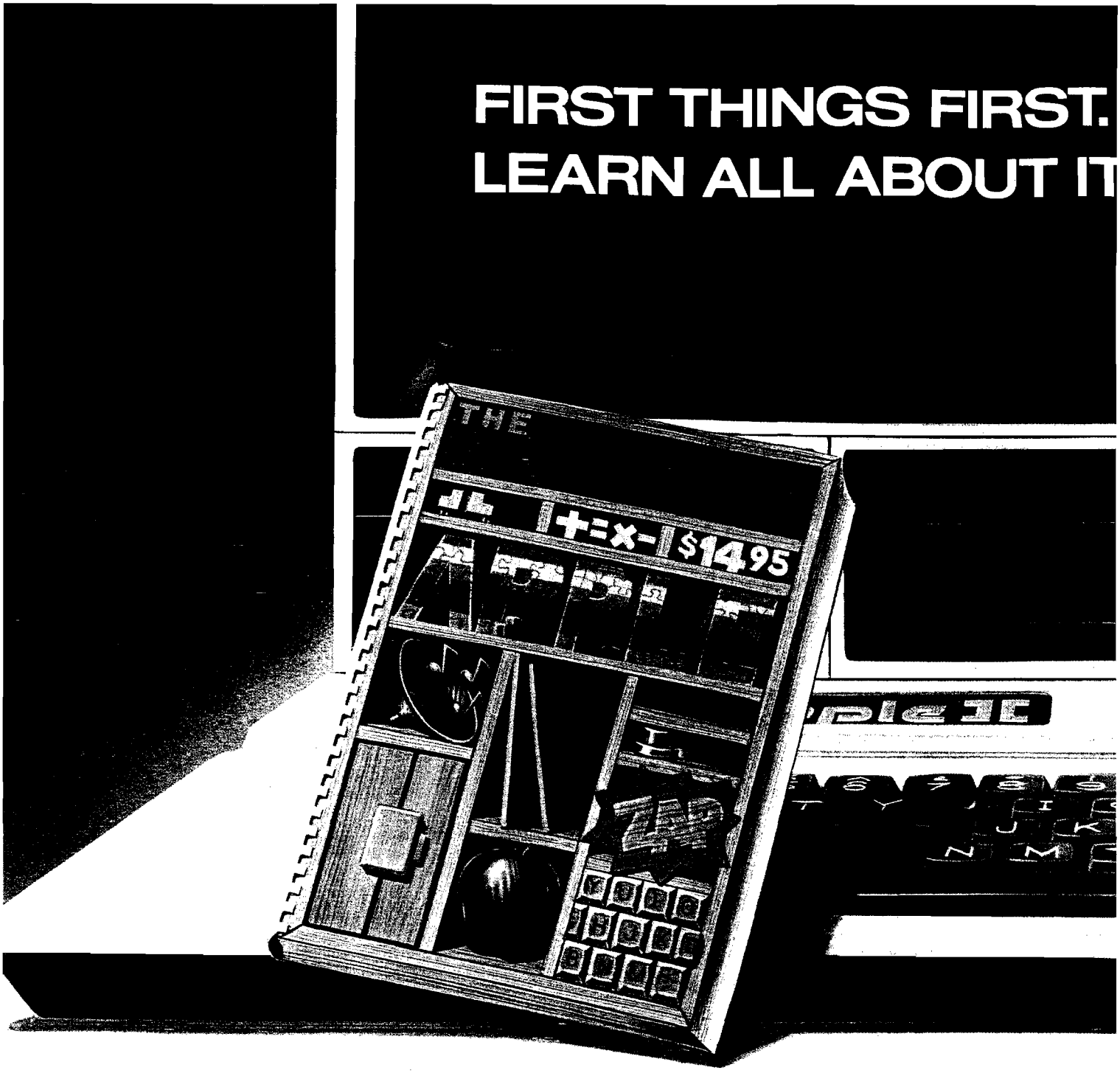
ATARI FORTH Screen Editor

APPLESOFT CAI Routine

PET Vet: Commodore 64's Super Sound

02
0 74470 11690 0

FIRST THINGS FIRST.
LEARN ALL ABOUT IT



When you don't know the first thing about your new Apple II* you need a friendly, cheerful, easy going teacher at your side. And the ELEMENTARY APPLE is just that kind of book.

It sweeps away the confusion—explains your Apple in everyday language—shows you how to hook it up, how to use the keyboard and work on the screen.

Gently and carefully it gives you an understanding of all the things your Apple can do. And then, it even shows how easy it is for anyone to write a simple program—provides common sense answers about graphics, utility programs, and the how and why of word processors, business programs and hardware like printers.

Yes, there's a lot of information. But, not one chapter o one word is dull or difficult to follow or complicated. Prove i yourself. Visit your computer store. Open the ELEMENTAR APPLE. Read a page of the introduction, then flip it open anywhere and read a paragraph or so. You'll find it's as understandable, as helpful and as marvelous as we say.

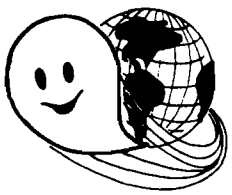
If you, or a member of your family, is an Apple beginn this is the book you need. It'll teach you everything you war know, in the way you want to learn.

Only $14.95. At computer and book stores, or:

DATAMOST INC

9748 Cozycroft Ave., Chatsworth, CA 91311. (213) 709-1202

VISA/MASTERCARD accepted. $2.00 shipping/handling charge.
(California residents add 6½% sales tax)
*Apple II is a trademark of Apple Computers, Inc.



2MHZ 6809 SYSTEMS

GIMIX offers you a variety to choose from!

38 MB WINCHESTER SYSTEM $17,498.99

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ 512KB Static RAM
- ★ 8 RS232C Serial Ports
- ★ 2 Parallel Ports
- ★ DMA Double Density Floppy Disk Controller
- ★ Dual 8" DSDD Floppy Disk System
- ★ Dual Winchester Subsystem with Two19 MB 5¼" Winchester Drives

SOFTWARE FEATURES:

- ★ OS-9 LEVEL TWO Multi-User Operating System
- ★ OS-9 Debugger
- ★ OS-9 Text Editor
- ★ OS-9 Assembler

19 MB WINCHESTER SYSTEM $8998.09

HARDWARE FEATURES:

- ★ 128K Static Ram
- ★ 2MHz 6809 CPU
- ★ 19 MB 5¼" Winchester DMA Subsystem
- ★ 4 RS232C Serial Ports
- ★ 1 MB 5¼" Floppy Disk Drive
- ★ DMA Double Density Floppy Disk Controller

SOFTWARE FEATURES:

- ★ OS-9 LEVEL TWO Multi-User Operating System
- ★ OS-9 Text Editor
- ★ OS-9 Debugger
- ★ OS-9 Assembler

128KB MULTI-USER SYSTEM $6997.39

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ DMA Double Density Floppy Disk Controller
- ★ 128KB Static Ram
- ★ 2 RS232C Serial Ports
- ★ Dual 8" DSDD Floppy Disk System

SOFTWARE FEATURES: Your choice of either UniFLEX or OS-9 LEVEL TWO. Both are Unix-like Multi-User/Multi-Tasking Operating Systems.

56KB FLEX/OS-9 "SWITCHING" SYSTEM $4148.49

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ 56K Static Ram
- ★ 2 RS232C Serial Ports
- ★ DMA Double Density Floppy Disk Controller
- ★ 2 Built-in 5¼" 40tr DSDD Disk Drives (80 Track DSDD Drive Option . . add $400.00)

SOFTWARE FEATURES:

- ★ GMXBUG monitor — FLEX Disk Operating System
- ★ OS-9 LEVEL ONE Multi-tasking operating system for up to 56K of memory

WINCHESTER SUBSYSTEMS

Winchester packages are available for upgrading current **GIMIX** 6809 systems equipped with DMA controllers, at least one floppy disk drive, and running FLEX, OS-9 LEVEL ONE or OS-9 LEVEL TWO. The packages include one or two 19MB (unformatted) Winchester drives, DMA Hard Disk Interface, and the appropriate software drivers. The Interface can handle two 5¼" Winchester Drives, providing Automatic Data Error Detection and Correction: up to 22 bit burst error detection and 11 bit burst error correction.

Dual drives can be used together to provide over 30 MBytes of on line storage -- or use one for back-up of the other. (More convenient and reliable than tape backup systems.

#90 includes one 19MB Drive, Interface, and Software **$4288.90**
#91 includes two 19MB Drives, Interface, and Software **$6688.91**

Contact GIMIX for systems customized to your needs or for more information.

50 HZ Export Versions Available

GIMIX Inc. reserves the right to change pricing and product specifications at any time without further notice.

GIMIX® and GHOST® are registered trademarks of GIMIX Inc.
FLEX and UniFLEX are trademarks of Technical Systems Consultants Inc.
OS-9 is a trademark of Microware Inc.

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510
TWX 910-221-4055

GIMIX inc.

1982 GIMIX Inc.

Emulates these terminals exactly.

IBM 3101
DEC VT100, VT52
Oata General D200
ADDS Regent 20, 25, 40
Hazeltine 1400, 1410, 1500
Lear Siegler ADM-3A, ADM-5
TeleVideo 910
Teletype Model 33 KSR

Apple is a trademark of Apple Computer, Inc.

SOFTERM®

High-speed CRT Look-alike Software for Your Apple®

New File Transfer Language

BREAK
CATALOG
CHAIN
CONFIGURE
CONNECT
CONVERSE
DIAL
END
HANGUP
LOG
MONITOR
NOLOG
ONERR
PAUSE
PROMPT
RECEIVE
REMARK
RETRIES
SEND
SPECIAL
SPEED
TIMEOUT
XMIT:WAIT

Supports these interface boards.

Apple Communications Card
Apple Parallel Printer
Apple Serial Interface
Apple Super Serial Card
Bit 3 Dual-Comm Plus™
CCS 7710,7720, 7728
Hayes Micromodem II™
Hayes Smartmodem™ 300, & 1200
Intra Computer PS10
Mountain Computer CPS Card™
Novation Apple-Cat II™ 300 & 1200
Orange Micro Grappler™
Prometheus VERSAcard™
SSM ASIO, APIO, AIO, AIO II™

Supports your 80-column hardware.

ALS Smarterm™
Bit 3 Full-View 80™
Computer Stop Omnivision™
M&R Sup'R'Terminal™
STB Systems STB-80™
Videx Videoterm™
Vista Computer Vision 80™
Wesper Micro Wizard 80™

Your host computer won't know the difference!

Softerm provides an *exact* terminal emulation for a wide range of CRT terminals which interface to a variety of host computer systems. Special function keys, sophisticated editing features, even local printer capabilities of the terminals emulated by Softerm are fully supported. Softerm operates with even the most discriminating host computer applications including video editors. And at speeds up to 9600 baud using either a direct connection or any standard modem.

Unmatched file transfer capability

Softerm offers file transfer methods flexible enough to match any host computer requirement. These include *character protocol* with user-definable terminator and acknowledge strings, block size, and character echo wait, and the intelligent *Softrans*™ protocol which provides reliable error-free transmission and reception of data. The character protocol provides maximum flexibility for text file transfers. *Any* type file may be transferred using the Softrans protocol which provides automatic binary encoding and decoding, block checking with error recovery, and data compression to enhance line utilization. A FORTRAN 77 source program is supplied with Softerm which is easily adaptable to any host computer to allow communications with Softerm using the Softrans protocol.

Softerm file transfer utilizes an easy to use *command language* which allows simple definition of even complex multiple-file transfers with handshaking. Twenty-three high-level commands include *DIAL, CATALOG, SEND, RECEIVE, ONERR, HANGUP, MONITOR* and others which may be executed in immediate command mode interactively or from a file transfer macro command file which has been previously entered and saved on disk.

Built-in utilities

Softerm disk utilities allow DOS commands such as CATALOG, INIT, RENAME, and DELETE to be executed allowing convenient file maintenance. Local file transfers allow files to be displayed, printed, or even copied to another file without exiting the Softerm program. Numerous editing options such as tab expansion and space compression are provided to allow easy reformatting of data to accommodate the variations in data formats used by host computers. Softerm supports automatic dialing in both terminal and file transfer modes. Dial utilities allow a *phone book* of frequently used numbers to be defined which are accessed by a user-assigned name and specify the serial interface parameters to be used.

Online Update Service

The Softronics Online Update Service is provided as an additional support service at no additional cost to Softerm users. Its purpose is to allow fast turnaround of Softerm program fixes for user-reported problems using the *automatic patch facility* included in Softerm as well as a convenient distribution method for additional terminal emulations and I/O drivers which become available. *User correspondence* can be electronically mailed to Softronics, and *user-contributed* keyboard macros, file transfer macros, and host adaptations of the Softrans FORTRAN 77 program are available on-line.

Most advanced communications software available

Just check Softerm's 300 page user manual. You simply can't buy a more sophisticated package or one that's easier to use. Available now for only $150 from your local dealer or Softronics, Inc.

SOFTRONICS

6626 Prince Edward, Memphis, TN 38119. 901-755-5006

MICRO™

February Highlights

Expanding your knowledge of computer languages — or just increasing your familiarity with many of them —can help improve your programming skills. This month's issue covers FORTH, Pascal, APL, and LISP, and provides you with a valuable information sheet of language packages offered by seventy vendors.

"EDIT, An Atari FORTH Screen-Oriented Editor" by Mike Dougherty (p. 47) uses the Atari 800 display as a text window into a FORTH disk screen. You can use Atari's special function keys to prepare FORTH applications. In "Apple Pascal Hi-Res Screen Dump" (p. 54), Robert Walker presents a high-resolution graphics screen dump for Apple Turtle graphics to the Epson printer with Graphtrax. Terry Peterson discusses the history and advantages of APL, a language known for its high execution speed, powerful features, yet cryptic character set — read "APL on the SuperPET" (p. 43). "The World According to LISP" by Steve Cherry (p. 65) is a good introduction to LISP, a language many computerists consider strange and obscure. Cherry outlines the major strengths and features of LISP, as well as its structure.

Commodore

In addition to the SuperPET article in our feature section, we have several PET articles and one on the VIC-20. "Microcomputer-Aided Instrumentation" (p. 89) by Deborah Graves, et. al., is a continuation of our series on Microcomputers in a College Teaching Laboratory (Part IV). Learn how to interface a microcomputer to two types of scientific instruments — a spectrophotometer and a chromatograph. If you want to convert Tiny PILOT to your 8K PET, read "More on Tiny PILOT" by Arthur Hunkins (p. 78). The author solves a few tricky problems and provides detailed explanations of some of the commands. David W. Priddle provides a utility program in "IEEE-488 Control of PET/CBM" (p. 11). You can add four new commands to your PET/CBM to make it an effective, inexpensive controller for use with many scientific instruments. The program requires 4.0 BASIC. And this month we present the third installment of Jim Strasma's "It's All Relative" (p. 33). Jim explains how to use the key file as an index into a relative file. He uses, as an example, a powerful mail-list package available to the public. Our VIC-20 contribution describes the hardware and software needed to interface RS-232 devices to Commodore's VIC computer. See Michael V. Tulloch's article, "An RS-232 Printer for VIC" (p. 17). You will also learn how to convert RS-232 voltages to TTL, convert hex code data to POKEs, and use a CTS line from the VIC.

Apple

Along with the Apple articles in our language feature, we offer an "Applesoft BASIC Routine for CAI" by Robert Phillips (p. 81). With this routine you can trap errors, isolate mistakes, and overlook typographical errors. Mr. Phillips also discusses some uses for a match routine, and presents just such a routine in BASIC.

Columns

This month's column on the Color Computer shows you how to interface your machine-language routines with BASIC (p. 92). John Steiner discusses the use of a RAM hook and presents a routine that interfaces with the LIST and LLIST commands to page a list on the screen. Tim Osborn presents BUILDIT (Apple Slices), a routine that makes programs external to VisiCalc create and access VisiCalc worksheet files (p. 95). Whether you are a beginner or pro, you can learn something from this program. Paul Swanson, in From Here to Atari (p. 31) discusses languages available for the Atari and answers several readers' questions on hardware. Commodore 64 fans will want to read Loren Wright's comments on the system's exciting sound capabilities (p. 71). He takes a look at music software, including *Synthy 64*, a musical composition program from Abacus Software.

MICRO™

NEW SECTION!

BEGINNING NEXT MONTH

Turn to page 112 for more information on our new section for the SERIOUS NOVICE appearing for the first time next month.

VIC BOOK!

Turn to PET Vet, page 73, for details on our soon-to-be-published book for the VIC.

Announcing
The best 6502 Assembler in the World

ORCA/M

Now. The kind of high-level support you'd only expect to find on a main frame. ORCA/M (*Hayden's Object Relocatable Code Assembler for Micros*) lets you develop sophisticated applications with the speed and ease of a high-level language, yet retain the control and efficiency that only assembly language can give.

Here's what ORCA/M gives you:

The Assembler

Macro language features:
- Conditional assembly of source and macro files
- Separate source and macro files
- Nestable macros
- Parameter mid-string and string search functions
- Symbolic parameter assignment
- Numeric, string, and boolean type parameters
- Parameter subscripting
- Global communication between macros
- Macro expansion loop control
- Count, length and type parameter-attribute functions

Extensive Macro Libraries

Memory Constant Declarations:
- Integer
- Character
- Four-byte Integer
- Hexadecimal
- Floating Point

Relocatable object module generation

Fast assembly directly to disk

Program segmentation:
- Selectively assemble individual subroutines
- Global and local scope of symbols

The Linker

Produce executable binary files from relocatable object modules

Link routines from library files

Link subroutine re-assemblies

Define a new origin for previously assembled code

Invoke at assembly time or by command

Subroutine libraries:
- Floating point and double precision routines
- Transcendental functions
- Hi- and lo-res graphics
- Multiple-precision integer math
- Input and output

The Editor

Co-resident screen editor:
- Global search and replace
- Block move
- Entry of non-keyboard characters

Supports lower case adapters and shift-key modification

80-column: horizontal scrolling with 40-column displays

The System

Monitor: transparent control of system from one command level

Extended Disk Commands:
- File copy
- File undelete
- Catalog sort
- Wildcard filenames

Disk ZAP: Built-in disk sector editor

Optimized DOS 3.3 compatible operating system

Operating system interface:
- Supports a variety of configurations
- User-modifiable to allow linkage of custom drivers for peripherals

64k RAM supported, 48k required

This unique array of features and functions speaks for itself: the power of **ORCA** is unsurpassed.

All features are documented clearly and extensively. Source listings for the subroutine and macro libraries, as well as the operating system, are included.

ORCA. If you're serious about developing 6502 software, it's the one to have.

Available from your local dealer, or call 800-343-1818. In MA call 617-937-0200. ORCA/M: 21609. Apple II disk, 48k, DOS 3.3. Two drives and 64k recommended.

Introductory Price: $99.95

HAYDEN SOFTWARE

NO. 57 FEBRUARY 1983

MICRO™

Advancing Computer Knowledge

STAFF

President/Editor-in-Chief
ROBERT M. TRIPP

Publisher
MARY GRACE SMITH

Editorial Staff
PHIL DALEY — Technical editor
JOHN HEDDERMAN — Jr. programmer
MARJORIE MORSE — Editor
JOAN WITHAM — Editorial assistant
LOREN WRIGHT — Technical editor

Graphics Department
HELEN BETZ — Director
PAULA M. KRAMER — Production mgr.
EMMALYN H. BENTLEY — Typesetter

Sales and Marketing
CATHI BLAND — Advertising manager
CAROL A. STARK — Circulation mgr.
LINDA HENSDILL — Dealer sales
MAUREEN DUBE — Promotion

Accounting Department
DONNA M. TRIPP — Comptroller
KAY COLLINS — Bookkeeper
EILEEN ENOS — Bookkeeper

Contributing Editors
CORNELIS BONGERS
DAVE MALMBERG
JOHN STEINER
JIM STRASMA
PAUL SWANSON
RICHARD VILE

Subscription/Dealer inquiries
(617) 256-5515

DEPARTMENTS

3 February Highlights
7 Editorial
9 Updates
31 From Here to ATARI
71 PET Vet
92 CoCo Bits
95 Apple Slices
98 Reviews in Brief
101 Software Catalog
103 Hardware Catalog
104 6809 Bibliography
105 Information Sheet
109 Data Sheet
111 Advertiser's Index
112 Next Month in MICRO

LANGUAGE FEATURE

42 **SuperPET APL** ..... *Terry Petersen*
An unusual, but powerful, fast, and memory-efficient language

47 **EDIT: An Atari FORTH Screen-Oriented Editor** ..... *Mike Dougherty*
A big improvement over the APX line editors

54 **APPLE Pascal Hi-Res Screen Dump** ..... *Robert D. Walker*
Dump the high-resolution graphics screen to your printer

58 **An Introduction to FORTH** ..... *Ronald W. Anderson*
All about Reverse Polish Notation, colon definitions, and other FORTH features

62 **FORTH for the 6809** ..... *Ronald W. Anderson*
A look at CCFORTH, figFORTH, and several FLEX-based systems

65 **The World According to LISP** ..... *Steven Cherry*
A powerful language suited to robotics and artificial intelligence

I/O ENHANCEMENTS

11 **Improved IEEE-488 Control for PET/CBM** ..... *David W. Priddle*
More precise control of the bus, without using logical files

17 **VIC RS-232 Printer** ..... *Michael V. Tulloch*
Interface the Radio Shack Quick Printer and other RS-232 devices

22 **PROM BASIC for the C1P** ..... *David A. Jones*
Increase the C1P's performance without a disk drive

28 **Indirect Files Under OS-65D** ..... *Richard L. Tretheway*
Use the indirect file to merge programs, perform warm starts, transfer programs.

BASIC AIDS

33 **It's All Relative, Part 3**
Using Commodore's Relative Records ..... *Jim Strasma*
Use a key file as an index into a relative file

37 **A Binary Search Routine** ..... *Alfred J. Bruey*
Two demonstration programs illustrate this technique

40 **BASIC Renumber for OSI** ..... *Paul Krieger*
Renumber programs in memory and save to tape

EDUCATIONAL UPDATES

74 **68000 Program Control: Branch and Jump Instructions** ..... *Joe Hootman*

78 **More on Tiny PILOT for the PET** ..... *Arthur Hunkins*

81 **A BASIC Match Routine for CAI** ..... *Robert Phillips*

86 **An Overview of Educational Software** ..... *George Gerhold*

89 **Microcomputers in a College Teaching Laboratory, Part 4** ..... *Deborah Graves, Richard H. Heist, Thor Olsen, Howard Saltsburg*

EVERYONE NEEDS A . . .

UNINTERRUPTABLE POWER SOURCE

ANGEL

SAVE YOUR DATA FROM POWER OUTAGES!

BACKUP FOR YOUR COMPUTER, MONITOR, PRINTER AND 5¼" FLOPPY AND HARD DISC DRIVE

• Automatically stops annoying problems from power line interruptions and brown outs • You need standby power to save data • Maintenance free backup power available in 115 volt or 220 volt • 50 or 60 HZ • 150 watts • Complete versatility — operate your system from a 12 volt source, i.e., automobile cigarette lighter, boat or airplane • Rugged self contained gel cell battery • No voiding warranty — no cutting wires • Automatic audio alarm warning tone during commercial power failure or interrupt • UL listed • FCC approved • Transient voltage suppressor gives added insurance from line voltage spikes, utilizing Zener Ray™ • Green/red LED power status indicator • Green — normal AC line power • Slow blinking red — at least 6 minutes of remaining standby power • Fast blinking red — approximately 2 minutes of remaining battery power • Solid state technology unexcelled by any UPS power unit in its class.

RH ELECTRONICS, INC.

COPYRIGHT © 1981 - PATENTS PENDING

566 IRELAN, BUELLTON, CA 93427

(805) 688-2047

SEE YOUR RH ELECTRONICS PRODUCTS DEALER

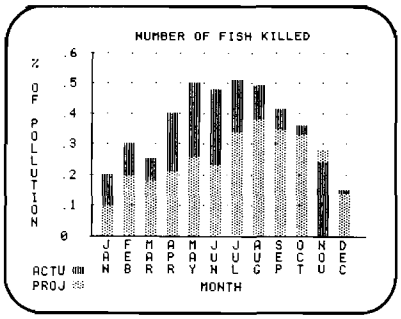
FOR YOUR APPLE II*:

SUPER FAN II™	$ 74.95
SUPER FAN II™/ZENER RAY™	$109.00
SUPER RAM II™	$125.00
RH 12 VOLT TRANSVERTER	$149.00

FOR MICRO COMPUTERS:

GUARDIAN ANGEL™	$595.00

About the Cover



Aquarium personnel now use microcomputers extensively for research, education, and cataloging data. The bar graph on this month's cover is a sample output from a computer used to monitor levels of toxic substances in ocean water. Other ways that microcomputers are being employed to improve the marine environment are as diverse as tracking whale migration patterns and feeding sites to regulating tank feedings with tidal rhythms.

Special thanks to the research department at Boston's New England Aquariaum for the time they spent with us discussing their work.

Cover photo: Phil Daley
Cover Graphic: Generated by program written by Art Arizpe

MICRO is published monthly by:
MICRO INK, Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to

MICRO INK
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

Copyright© 1982 by MICRO INK
All Rights Reserved

MICRO™ Editorial

Our language feature this month provides an appropriate forum for our Editor-in-Chief Bob Tripp, and Technical Editor Phil Daley, to express their opposing views on BASIC. It won't take long to figure out who is pro and who is con. What are *your* thoughts on BASIC?

Too Basic or Not Too Basic

According to John Kemeny, one of the originators of BASIC, the main intent was to provide the user with friendly access to the computer. This emphasis on user friendliness is the key to BASIC's popularity. The key word in Beginner's All-purpose Symbolic Instruction Code is the first one; more microcomputer programmers start with BASIC than any other language.

Without BASIC the microcomputer world would never have gotten off the ground. Its simplicity, forthright clarity, memorable mnemonics, and interactive friendliness combine to make it the best all-round microcomputer programming language. When the micro had only 4K of ROM and 2K of RAM, BASIC was a necessity; even now with 64K- and 128K-RAM machines, loading a monolith language returns you to limited memory constraints.

Other languages, such as FORTRAN and COBOL, make it easy to do programming and to understand the program. Unfortunately, microcomputers don't do them justice, and all implementations become subsets of the mother tongue.

FORTH and assembly language overcome many of the limitations of BASIC — especially its slowness — but they are incomprehensible jibberish to most folks, sometimes including the person who wrote the program.

I must also mention Pascal, a fine structured language, but the most exasperating language in which to write a program. The disk accesses, even with three drives, are incredible. The routine — load the editor, load the file, change the file, save the file, load the compiler, compile the program, run the program, note the mistakes, load the editor — is enough to addle your brains. Any program that knows enough to tell you that you forgot a semicolon on the previous line, should be smart enough to insert one for you!

Give me BASIC any day: load the program, run, and make changes with no delays. It is the only user-friendly language amongst the lot!

Phil Daley

Much Too Basic and Too Much BASIC!

BASIC was designed to allow a student with a TTY terminal to write simple programs on a time-shared mainframe, and thousands of people received their initial introduction to the computer *via* BASIC. It served these purposes well. However, it is not a good language for the microcomputer. The use and abuse of BASIC has caused software development to lag behind hardware development. The fundamental problems are:

1. BASIC does not make effective use of screen capabilities. Positioning the cursor is awkward, reading the cursor position is difficult, and material appearing on the display is virtually impossible to access from a program.

2. Input routines do not permit error checking of input, do not support interaction with the operator, do not provide support for other input devices, and are generally limited in capability.

3. BASIC does not make effective use of disk capacities. Only the most basic disk file structures are supported, and these often have restrictions.

4. BASIC does not support large programs well. The use of line numbers as labels makes it difficult to maintain and modify large programs. As programs grow, they become increasingly hard to modify, document, and understand.

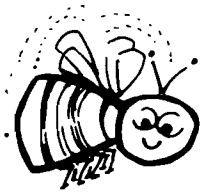
5. BASIC does not have simple techniques for manipulating string-oriented material. It 'thinks' only in decimal, which may be appropriate for some calculations, but is awkward for many requirements of hexadecimal-oriented computers.

6. BASIC is slow. The overhead involved with interpreting and reinterpreting every line during program execution can make even the most efficient microcomputers appear sluggish. There is also a high overhead in its memory requirements.

BASIC is simply too basic for many situations. Some programs would be more beneficial written in an alternative language. BASIC should be used for introductory programming, short programs, numerical calculation programs, and limited input requirements. It should not be used for long programs, disk and video-oriented applications, non-numerical programs, and special devices; in these cases, alternative approaches should be considered.

Bob Tripp

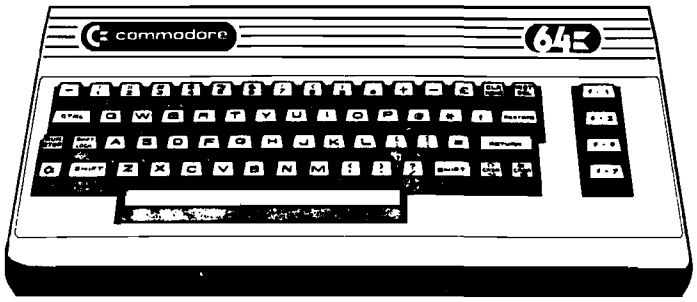
BUSICALC



BUSICALC A Honey of an Electronic Spreadsheet

Why electronic spreadsheet programs?

Electronic spreadsheet programs allow the user to create a gridsheet, spreadsheet, worksheet, or any other table of information, using the memory of the computer as pencil and paper. The computer display or terminal acts as a window through which the user views the information as it is entered. Textual information (such as headings), numerical values, and formulas can easily be entered into the spreadsheet.



For Commodore 64
For Commodore VIC 20
For Commodore PET/CBM 40 columns
For Commodore CBM 80 column/SuperPet

BUSICALC Your Computer Drone for Repetitive Calculations

The outstanding advantage of using a computer is that it acts not only as a pencil and paper but as a perfect eraser and an automatic calculator. The user can quickly and easily make any number of alterations to the data within the table. The BUSICALC will evaluate any formula using the data that has been entered. Further, it retains the formulas and displays the resulting value. With BUSICALC controlling the entry of data, providing a comprehensive memory, and performing arithmetic, the preparation of a spreadsheet is faster and more accurate than if it were prepared by hand.

BUSICALC With the Sting Removed from the Prices

BUSICALC 20. . . . . only **$49.00** for the VIC 20
BUSICALC 64. . . . . only **$69.00** for the CBM 64
BUSICALC 40. . . . . only **$79.00** for the original 40 column PET/CBM
BUSICALC 80. . . . . only **$89.00** for the original 80 column CBMs and SuperPets

BUSICALC AVAILABLE NOW FROM YOUR LOCAL DEALER
(800) 227-9998
FOR THE NAME OF YOUR NEAREST DEALER

California, Canada, Alaska and Hawaii please call (415) 965-1735

Skyles Electric Works
231G South Whisman Road
Mountain View, CA 94041

Europe please contact Supersoft, Winchester House, Canning Road, Harrow Wealdstone, Middlesex, England HA3 7SJ, Tel. 01 861 1166

Updates and Microbes

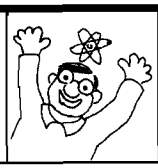
In MICRO's Commodore 64 Data Sheet [MICRO 55:109] most of the SID's registers were inadvertently omitted. Here is a complete SID register list.

Voice 1	Voice 2	Voice 3	Register Function
$D400 (54272)	$D407 (54279)	$D40E (54286)	Frequency, low byte
$D401 (54273)	$D408 (54280)	$D40F (54287)	Frequency, high byte
$D402 (54274)	$D409 (54281)	$D410 (54288)	Pulse width, low byte
$D403 (54275)	$D40A (54282)	$D411 (54289)	Pulse width, high nibble (bits 4-7 = 0)
$D404 (54276)	$D40B (54283)	$D412 (54290)	Voice Type: (bits 4-7) 7 Noise 6 Pulse 5 Sawtooth 4 Triangle 3 Test 2 Ring modulate (1 = on, 0 = off) 1 Synchronize 0 Gate bit (1 = start attack, 0 = start release)
$D405 (54277)	$D40C (54284)	$D413 (54291)	Attack/Decay Attack time (bits 4-7) 2 ms - 8 ms Decay time (bits 0-3) 6 ms - 24 sec
$D406 (54278)	$D40D (54285)	$D414 (54292)	Sustain/Release Sustain level (bits 4-7) x/15 proportion of peak Release time 6 ms - 24 sec

Address		Register Function
$D415	54293	Filter frequency Bits 7-3 = 0 Bits 2-0 low bits
$D416	54294	Filter frequency, high byte
$D417	54295	Resonance/Filter Voices 7-4 Resonance 3-0 Filter voices 3 external 2 voice 3 1 voice 2 0 voice 1
$D418	54296	Filter select/Master volume 7 voice 3 off 6 high-pass on 5 band-pass on 4 low-pass on 3-0 master volume
$D419	54297	Paddle X (A/D #1)--read only
$D41A	54298	Paddle Y (A/D #2)--read only
$D41B	54299	Digitized voice 3 waveform--read only
$D41C	54300	Digitized voice 3 envelope--read only

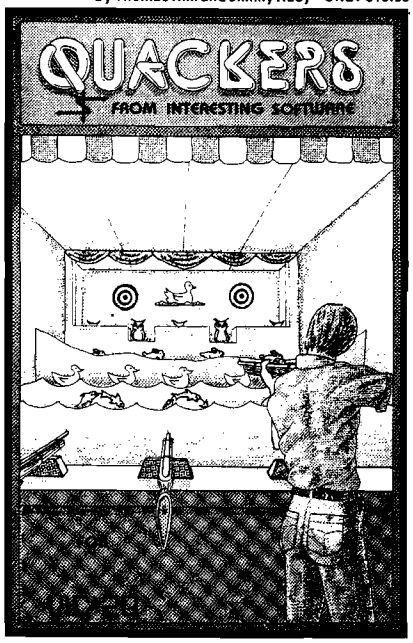
Let us know if you've updated an atricle or discovered a bug. Send a note to: Updates/Microbes, MICRO, P.O. Box 6502, Chelmsford, MA 01824. MICRO

VIC-20* SOFTWARE SPECIALS VIC-20*

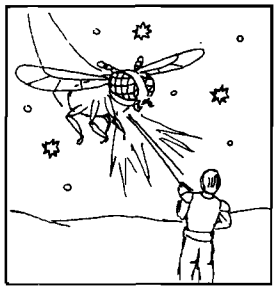


NEW

A 100% machine code game with colorful graphics, music, sound and a funny looking turtle to entertain your entire family! By Thomas Kim and Jimmy Huey ONLY $15.95



FROM TRONIX



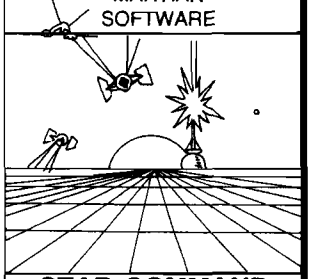
SWARM!

Another fast action game written entirely in machine language from Tronix. Insects invade your Vic!

Cassette $29.95

DUST COVERS - $7.95
For Vic-20 or Vic-64
* Waterproof
* Brown Color
* Commodore Logo
Protect your investment!

FROM MARTIAN SOFTWARE



STAR COMMAND
* Intergalactic Combat! * Space Conflict
* All Machine Language
Cassette $16.95

INTERESTING SOFTWARE
21101 S. Harvard Blvd., Torrance, CA 90501
(213) 328-9422

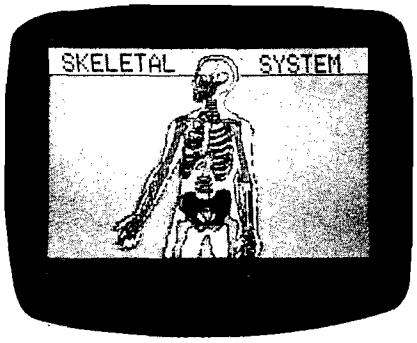
Visa/MC/Check/Money Order
CA residents add appropriate sales tax
Write for free Catalog
Add $2.00 Postage & Handling
Dealer Inquirers Invited

*Vic-20 is a trademark of Commodore Business Machines

FOR COMPLETE GRAPHICS:

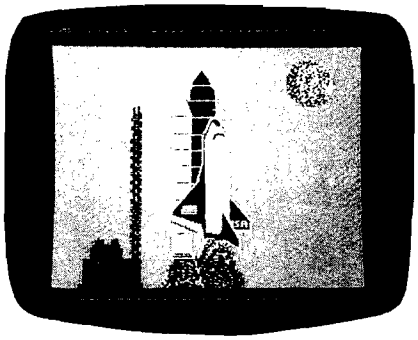
VersaWriter

EDUCATION

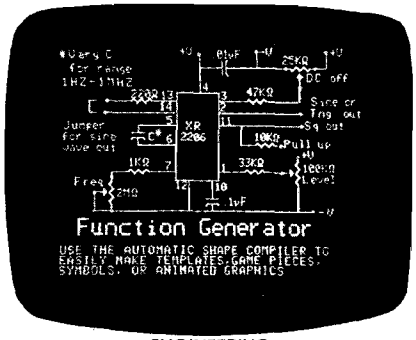


ARTIST

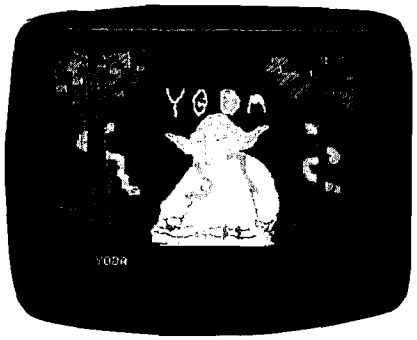
GAME PROGRAMMER



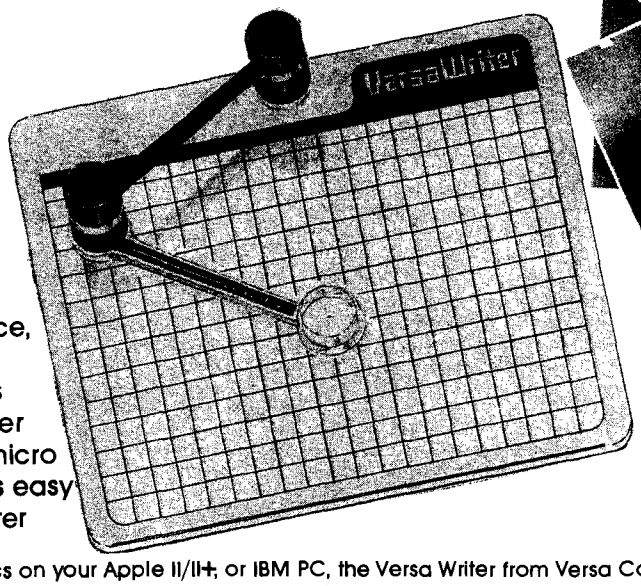
HOBBIEST

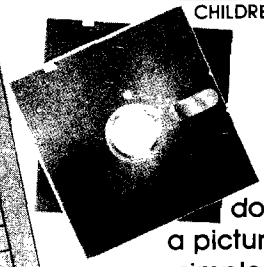

Function Generator
USE THE AUTOMATIC SHAPE COMPILER TO EASILY MAKE TEMPLATES,GAME PIECES, SYMBOLS, OR ANIMATED GRAPHICS

ENGINEERING



CHILDREN





Whether you are a teacher, an artist, an engineer, a programmer, or a hobbiest with little or no programming experience, the VersaWriter is the answer to your graphics need. With the VersaWriter exploring the world of micro computer graphics is as easy as tracing. The VersaWriter doesn't just trace a picture though. With simple one key commands you can add color and text to your pictures, move objects across the screen, make scale drawings and even draw with different size brushes. The VersaWriter is as limitless as your imagination.

For complete graphics on your Apple II/II+, or IBM PC, the Versa Writer from Versa Computing, Inc. is your answer. Complete hardware/software system ready for use - $299.

Send for information on the complete line of VERSAWARE & Graphics Products

VERSA COMPUTING, INC.

Dealer Inquiries Welcome

3541 Old Conejo Road, Suite 104 • Newbury Park, CA 91320 • (805) 498-1956

Improved IEEE-488 Control for PET/CBM

by David W. Priddle

This utility program for PET/CBM with 4.0 BASIC adds four new commands that improve the I/O control of the IEEE-488 bus. With more complete control of the bus, the PET/CBM becomes an effective, inexpensive controller for a wide variety of scientific instruments.

I/O Control
requires:
4.0 PET/CBM or SuperPET

The IEEE-488 bus is available on a wide variety of measurement and signal-generating equipment from a number of large manufacturers including Hewlett-Packard, Tektronix, and Fluke. Some of these companies also supply sophisticated and expensive microcomputers and "programmable calculators" that can act as instrument controllers using the 488 bus. The PET/CBM micros cost considerably less and are, therefore, an attractive alternative for use with scientific instruments. The disadvantages of the PET/CBMs are twofold: the graphics resolution is poor when compared to a Tektronix 4051, which has a 1024 × 780 point addressable display; and second, the I/O techniques using "logical files" have limitations when used with equipment other than disk drives and printers.

The resolution problem has been tackled in a variety of ways, from simple plotting programs that offer a small improvement, to the addition of expansion graphics memory. The problem of improving the I/O control is the object of this current utility program.

The Tektronix 4051/4052 systems offer two levels of control. High-level control of the bus uses the commands:

PRINT@X,Y:a,b,c,etc,
INPUT@X,Y:a,b,c,etc,

***Listing 1:* Assembly Listing**

```
0001  0000             ;'print@27'
0002  0000             ;aug 26,1982
0003  0000                    *=$0030
0004  0030             ;start of strings
0005  0030                    *=$0034
0006  0034             ;top of basic
0007  0034                    *=$0070
0008  0070             chrget                 ;jump to start
0009  0070                    *=$0076
0010  0076             chrgot
0011  0076                    *=$027a
0012  027a             temp3  *=*+1           ;y store
0013  027b             temp4  *=*+1           ;i/o flag,0=output,$22=input
0014  027c             temp5  *=*+1           ;rby buffer
0015  027d             temp6  *=*+1           ;quote delimiter
0016  027e             temp7  *=*+1           ;colon delimiter
0017  027f                    *=$7e47
0018  7e47  a9 7e       setup  lda #$7e        ;sys 32327 to setup
0019  7e49  85 31              sta $31         ;change top of strings
0020  7e4b  85 35              sta $35         ;and top of memory
0021  7e4d  a9 46              lda #$46
0022  7e4f  85 30              sta $30
0023  7e51  85 34              sta $34
0024  7e53  a9 4c              lda #$4c        ;put jmp ($4c) in chrget
0025  7e55  85 70              sta $70
0026  7e57  a9 e5              lda #$e5        ;adjust to point at start
0027  7e59  85 71              sta $71
0028  7e5b  a9 7e              lda #$7e
0029  7e5d  85 72              sta $72
0030  7e5f  60                 rts
0031  7e60  20 70 00    input4 jsr chrget      ;go past delimiter
0032  7e63  a2 00       input  ldx #$00
0033  7e65  20 c0 f1    input2 jsr $f1c0       ;get byte from ieee
0034  7e68  9d 7c 02           sta $027c,x     ;store byte in buffer
0035  7e6b  e8                 inx
0036  7e6c  24 10              bit $10         ;check,was '\' used ?
0037  7e6e  30 04              bmi input5      ;'\' so don't check for cr
0038  7e70  c9 0d              cmp #$0d        ;check for cr
0039  7e72  f0 0a              beq input3      ;cr, so end input
0040  7e74  e0 fa       input5 cpx #250        ;check if full
0041  7e76  f0 06              beq input3      ;buffer full so quit
0042  7e78  a5 96              lda $96         ;status word
0043  7e7a  29 40              and #%01000000  ;check eoi
0044  7e7c  f0 e7              beq input2      ;no eoi so continue
0045  7e7e  a9 22       input3 lda #$22        ;quote
0046  7e80  9d 7c 02           sta $027c,x
0047  7e83  e8                 inx
0048  7e84  a9 3a              lda #$3a        ;colon
0049  7e86  9d 7c 02           sta $027c,x

0051  7e89  20 d1 7f           jsr varval      ;assign value
0052  7e8c  20 76 00           jsr chrgot      ;check for 0
0053  7e8f  d0 cf              bne input4
0054  7e91  4c df 7e           jmp print3      ;input finished

0056  7e94  a9 00       inout  lda #$00        ;set status word=0
0057  7e96  85 96              sta $0096
0058  7e98  20 d1 c8    print  jsr $c8d1       ;get byte to .xr
0059  7e9b  86 d4              stx $d4         ;pa
0060  7e9d  86 b0              stx $b0         ;output device number
0061  7e9f  86 af              stx $af         ;input device number
0062  7ea1  ad 7b 02           lda temp4       ;check i/o flag
0063  7ea4  f0 07              beq inout2      ;output, send listen
0064  7ea6  8a                 txa             ;put pa into .a
0065  7ea7  20 d2 f0           jsr $f0d2       ;talk addr routine
0066  7eaa  18                 clc
0067  7eab  90 04              bcc print2      ;branch always
0068  7ead  8a          inout2 txa             ;put pa into .a
0069  7eae  20 d5 f0           jsr $f0d5       ;listen addr routine
0070  7eb1  20 76 00    print2 jsr chrgot      ;check for semicolon
0071  7eb4  c9 3b              cmp #$3b
```

where X is the IEEE-488 device primary address (PA), Y is the secondary address (SA), and a, b, and c are the variables to be output or input. The utility program presented here follows this syntax with the single change of using a semicolon or a backslash in place of the colon. (The PET/CBM recognizes the colon as a statement separator.) This syntax has the obvious advantage that "logical files" are not used: there is no need to "OPEN" or "CLOSE" files. The output has the following syntax:

```
PRINT@4,0;"hello there",a,b%,c$
```

This command string sends to the 488 bus the primary listen address 4 and the secondary address 0 (with the 488 ATN line set low — true), followed by the ASCII string "hello there", the current values of the variable a, integer variable b%, and string variable c$. A carriage return is sent and the 488 bus is cleared by an UNLISTEN/UNTALK command.

If a backslash is used in place of the semicolon, a line feed is also sent with the carriage return. If a printer with device number 4 is connected, the string and variables will be printed.

The input command has the syntax:

```
INPUT@29,24;a$
```

This command string sends the primary talk address 29 and the secondary address 24. Data is then input to the PET/CBM until a carriage return is received, the 488 EOI (end or identify) line is set low (= true), or the input buffer is full. The data is then assigned to the variable a$. There are three features of this syntax that differ from the standard PET/CBM INPUT#: no logical files are used; the input buffer used is located in the tape buffers and is set to allow an input of 250 characters instead of the usual limit of 80 characters; and the command can be executed not only in a program, but in immediate mode. This feature is useful when controlling instruments. In addition, if a backslash is used in place of the semicolon, the input will not terminate when a carriage return is received. Any carriage returns will simply be imbedded in the string variable (in the above case, a$). Of course, numeric variables can also be input with INPUT@. In these cases the backslash should not be used since the carriage return is an illegal character here.

The Tektronix systems also allow

Listing 1 *(continued)*

```
0072  7eb6  f0 0e        beq atnhi2
0073  7eb8  c9 5c        cmp #$5c        ;check for \
0074  7eba  f0 0a        beq atnhi2
0075  7ebc  20 d1 c8     jsr $c8d1       ;get byte (sa) to .xr
0076  7ebf  86 d3        stx $d3         ;sa
0077  7ec1  86 a5        stx $a5         ;sa into ieee buffer
0078  7ec3  20 09 f1     jsr $f109       ;send sa
0079  7ec6  20 48 f1  atnhi2 jsr $f148   ;clear atn (high)
0080  7ec9  20 76 00     jsr chrgot
0081  7ecc  c9 5c        cmp #$5c        ;check for \
0082  7ece  d0 04        bne print4      ;send cr,end input on cr
0083  7ed0  a9 ff        lda #$ff        ;send cr/lf,ignore cr on inp
0084  7ed2  85 10        sta $10
0085  7ed4  ad 7b 02  print4 lda temp4   ;check if input
0086  7ed7  d0 87        bne input4      ;input data
0087  7ed9  20 70 00     jsr chrget

0089  7edc  20 aa ba     jsr $baaa       ;send data
0090  7edf  20 b4 bb  print3 jsr $bbb4   ;unl buss,reset i/o
0091  7ee2  4c fa 7e     jmp bacbas      ;return to basic

0093  7ee5  8c 7a 02  start  sty temp3   ;save y
0094  7ee8  e6 77        inc $77
0095  7eea  d0 02     bl   bne b2
0096  7eec  e6 78        inc $78
0097  7eee  a0 00     b2   ldy #$00
0098  7ef0  b1 77        lda ($77),y
0099  7ef2  c9 99        cmp #$99        ;check for print
0100  7ef4  f0 0a        beq b5
0101  7ef6  c9 85        cmp #$85        ;check for input
0102  7ef8  f0 09        beq a1
0103  7efa  ac 7a 02  bacbas ldy temp3   ;restore y
0104  7efd  4c 76 00     jmp chrgot      ;back to basic

0106  7f00  a9 00     b5   lda #$00      ;set output flag
0107  7f02  2c           .byte $2c       ;dummy op (bit = $2c)
0108  7f03  a9 22     a1   lda #$22      ;set input flag (")
0109  7f05  8d 7b 02     sta temp4       ;store flag
0110  7f08  20 70 00     jsr chrget

0111  7f0b  c9 40        cmp #$40        ;check for @
0112  7f0d  f0 85        beq inout
0113  7f0f  c9 21        cmp #$21        ;check for !
0114  7f11  f0 06        beq rbywby
0115  7f13  20 f7 7f     jsr chrdec      ;reset chrget pointer
0116  7f16  4c fa 7e     jmp bacbas
0117  7f19  ad 7b 02  rbywby lda temp4   ;check i/o flag
0118  7f1c  d0 6e        bne rby

0120  7f1e  20 70 00  wby    jsr chrget
0121  7f21  c9 3b        cmp #$3b        ;check for semicolon
0122  7f23  f0 31        beq out2        ;no addr sequence
0123  7f25  20 f7 7f     jsr chrdec
0124  7f28  20 d1 c8     jsr $c8d1       ;get byte to .xr
0125  7f2b  86 d4        stx $d4         ;prim addr save
0126  7f2d  a9 00        lda #$00        ;send pa "as is"
0127  7f2f  20 d7 f0     jsr $f0d7       ;set atn, send pa

0129  7f32  20 76 00     jsr chrgot      ;check for semicolon
0130  7f35  c9 3b        cmp #$3b
0131  7f37  f0 0a        beq atnhi       ;no s.a. so clear atn
0132  7f39  20 d1 c8     jsr $c8d1       ;get byte (sa) to .xr
0133  7f3c  86 d3        stx $d3         ;sa store
0134  7f3e  86 a5        stx $a5         ;ieee output buffer
0135  7f40  20 09 f1     jsr $f109       ;send byte to ieee
0136  7f43  a9 fd     atnhi  lda #$fd    ;set nrfd low
0137  7f45  2d 40 e8     and $e840
0138  7f48  8d 40 e8     sta $e840
0139  7f4b  20 48 f1     jsr $f148       ;atn off (high)
0140  7f4e  20 70 00     jsr chrget      ;look past semicolon
0141  7f51            ;z flag set if binary zero or colon
0142  7f51  f0 a7        beq bacbas      ;no bytes to send,bacbas
0143  7f53  20 f7 7f     jsr chrdec
0144  7f56  a9 02     out2   lda #$02    ;set nrfd high again
0145  7f58  0d 40 e8     ora $e840
0146  7f5b  0d 40 e8     sta $e840

0148  7f5e  20 76 00  outbyt jsr chrgot
0149  7f61  f0 97        beq bacbas      ;check for end of statement
0150  7f63  20 70 00     jsr chrget      ;look at next chr
0151  7f66  c9 ab        cmp #$ab        ;check for minus sign
0152  7f68  f0 0e        beq out3
0153  7f6a  20 f7 7f     jsr chrdec      ;reset chrget
0154  7f6d  20 d1 c8     jsr $c8d1       ;get byte to .xr
0155  7f70  86 a5        stx $a5         ;to ieee buffer
0156  7f72  20 09 f1     jsr $f109       ;send to ieee
0157  7f75  4c 5e 7f     jmp outbyt

0159  7f78  a9 34     out3   lda #%00110100  ;bit 3
0160  7f7a  8d 11 e8     sta $e811       ;set eoi true
0161  7f7d  20 d1 c8     jsr $c8d1       ;get byte
0162  7f80  86 a5        stx $a5
0163  7f82  20 09 f1     jsr $f109       ;send it
0164  7f85  a9 3c        lda #%00111100  ;bit 3
0165  7f87  8d 11 e8     sta $e811       ;reset eoi
```

Listing 1 *(continued)*

```
0166  7f8a  10 d2           bpl outbyt      ;branch always

0168  7f8c  a9 22     rby   lda #$22        ;set up buffer
0169  7f8e  8d 7d 02        sta temp6
0170  7f91  a9 3a           lda #$3a
0171  7f93  8d 7e 02        sta temp7
0172  7f96  20 70 00        jsr $0070       ;go past (!)
0173  7f99  20 c0 f1        jsr $f1c0       ;get byte from ieee
0174  7f9c  8d 7c 02        sta temp5       ;get unassigned byte
0175  7f9f  85 a5           sta $a5         ;copy also to ieee buffer
0176  7fa1  20 76 00        jsr chrgot      ;check chrgot
0177  7fa4  d0 15           bne rby4        ;no target variable
0178  7fa6  a9 22     rby5  lda #$22        ;defeat housekeeping
0179  7fa8  8d 7d 02        sta temp6
0180  7fab  a9 3a           lda #$3a
0181  7fad  8d 7e 02        sta temp7
0182  7fb0  a9 08     rby3  lda #$08        ;set ndac high
0183  7fb2  0d 21 e8        ora $e821
0184  7fb5  8d 21 e8        sta $e821
0185  7fb8  4c fa 7e        jmp bacbas

0187  7fbb  20 d1 7f  rby4  jsr varval
0188  7fbe  20 76 00        jsr chrgot      ;check for 0
0189  7fc1  f0 e3           beq rby5
0190  7fc3  20 70 00        jsr chrget      ;go past delimiter
0191  7fc6  20 c0 f1        jsr $f1c0       ;get next byte
0192  7fc9  8d 7c 02        sta temp5       ;store
0193  7fcc  85 a5           sta $a5
0194  7fce  4c bb 7f        jmp rby4        ;assign value

0196  7fd1  20 2b c1  varval jsr $c12b      ;search for variable
0197  7fd4  85 46           sta $46         ;returns in .a and .yr
0198  7fd6  84 47           sty $47         ;variable pointers
0199  7fd8  a5 77           lda $77
0200  7fda  48              pha
0201  7fdb  a5 78           lda $78
0202  7fdd  48              pha             ;save chrget pointers

0204  7fde  24 07           bit $07         ;var type, ff=str 00=numeric
0205  7fe0  30 03           bmi rby2        ;str if true,n flag set by bit
0206  7fe2  a9 7c           lda #$7c
0207  7fe4  2c              .byte $2c       ;dummy op code
0208  7fe5  a9 7b     rby2  lda #$7b        ;start buffer at quote
0209  7fe7  85 77           sta $77
0210  7fe9  a9 02           lda #$02        ;change chrget pointers
0211  7feb  85 78           sta $78
0212  7fed           ;above values must change if buffer moved

0214  7fed  20 3c b9        jsr $b93c       ;assign var value
0215  7ff0  68              pla             ;reset chrget
0216  7ff1  85 78           sta $78
0217  7ff3  68              pla
0218  7ff4  85 77           sta $77
0219  7ff6  60              rts
0218  7ff4  85 77           sta $77
0219  7ff6  60              rts


0222  7ff7  a5 77     chrdec lda $77        ;reset chrget pointer
0223  7ff9  d0 02           bne chrd1
0224  7ffb  c6 78           dec $78
0225  7ffd  c6 77     chrd1 dec $77
0226  7fff  60              rts
```

488 bus control on a more primitive level using the syntax:

WBYTE@X,Y:a,b,−c
RBYTEa,b

The WBYTE command means "write-byte" and is implemented in this program with the syntax:

PRINT!X,Y;a,b,−c

For those familiar with Tektronix, this command behaves exactly as WBYTE. The command sends (with ATN set) the absolute primary address of the value of X, and the absolute secondary address of the value of Y. By "absolute," I mean that the address is sent as is and is not first converted to a talk address (by setting bit 7), or a listen address (by setting bit 6). The byte variables a, b, and c are then sent (without ATN). These one-byte variables (or expressions) must have values between 0 and 255 to be valid. The minus sign before the variable c causes the EOI line to be set as this byte is sent. Depending on the particular device, this may be necessary to signal the end of transmission to the addressed device.

Note that the command does not send UNLISTEN or UNTALK and the addressed device(s) continues to take part in transactions on the 488 bus. This means that one talker and one or more listeners can be set actively on the bus with or without the controller (the PET/CBM) taking part in the subsequent data transactions. When the process is finished, it may be necessary to send UNL/UNT using the command:

PRINT!63,95;

This will force all previously addressed talkers and listeners off the 488 bus.

The Tektronix RBYTE command means "readbyte" and is exactly implemented here using the syntax:

INPUT!a,b

The use of this command requires that a 488 bus device must have been previously addressed as a talker (using PRINT!(PA),(SA);). Execution of the command causes the talker to send a single byte whose ASCII value is then assigned by the PET/CBM to the variable a, and then to send another single byte, which will be assigned to b. Any number or types of variables may be specified and a single byte will be input for each. If no variable is specified, an unassigned byte will be input and placed in location ($00A5) where it may be PEEKed if desired. Again, the device must be sent UNTALK (PRINT!95;) in order to remove it from the bus.

This command is unlike GET# in two important respects: again, no logical files are used and INPUT! does not send either an address sequence or an UNTALK command. The GET# command does not allow a talker to stay actively on the bus because UNTALK is sent during each execution. GET# goes through the addressing routine each time before it gets a byte from the 488 bus. These two aspects of GET# can be merely an inconvenience, or an absolute disaster, when attempting to use the PET/CBM to control advanced instrumentation. With one instrument we have used, GET# caused the PET/CBM to receive only every third byte sent because of the confusion caused by the repeated addressing sequences. INPUT# could not be used since more than 80 characters are sent without either EOI or a carriage return.

This utility program mimics the I/O procedures used by Tektronix and thus allows the PET/CBM to send or receive data using the simpler syntax of PRINT@; and INPUT@; without

needing "logical files." It also permits any single byte to be sent or received (with or without ATN) on the 488 bus using the syntax of PRINT!; and INPUT! . More complete control of I/O on the IEEE-488 bus allows the PET/CBM to be used as a scientific instrument controller in applications that are more difficult or impossible without this program.

How to Use the PRINT@ Utility Program

An assembler listing (listing 1) and BASIC loader program (listing 2) are provided for any PET/CBM with BASIC 4.0 and 32K. The BASIC program should be entered and run. This will load the machine code into top of memory, adjust the BASIC pointers to protect the code, and attach itself to the CHRGET routine. The utility program may then be NEWed. Since the machine code in this form is not relocatable, it should be loaded before any other BASIC program. There are 12 absolute addresses used that must be changed if the program is reassembled for some other location such as in an EPROM.

The four new commands are implemented using some of the PET/CBM ROM routines, but it was not possible to use the jump address table in ROM since only parts of the I/O routines are used. The commands support the full syntax error checking for variable type and legal values, as well as for punctuation.

When the commands are entered, BASIC crunches the PRINT or INPUT to the usual tokens ($99 or$85). When BASIC executes the commands, the utility program tests to see if a PRINT or INPUT has been found. If either is found then a further test is made to see if they are followed by @ or ! and the correct routine is then started. If neither test succeeds, then control is sent back to BASIC.

(Editor's note: this approach slows down the execution of BASIC programs to varying degrees, depending on the particular instructions involved. If you don't need to use the commands for a particular program, it is probably best to turn the machine off before loading a new BASIC program.

COMPU SENSE

CARDBOARD 6
$87.95
An expansion interface for the VIC-20. Allows expansion to 40 K or accepts up to six games. May be daisy chained for more versatility.

CARDBOARD 3
$39.95
Economy expansion interface for the VIC-20

CARD "?" CARD/PRINT
$79.95
Universal Centronics Parallel Printer Interface for the VIC-20 or CBM-64. Use an Epson MX-80 or OKIDATA or TANDY or just about any other.

CARDETTE
$39.95
Use any standard cassette player/recorder with your VIC-20 or CBM-64

CARDRITER
$39.95
A light pen with six good programs to use with your VIC-20 or CBM-64

Prices subject to change.
TO ORDER: P.O. BOX 18765
WICHITA, KS 67218
(316) 684-4660
Personal Checks Accepted (Allow 3 Weeks)
or C.O.D. (Add $2) Handling Charges $2.00

***Listing 2:* BASIC Loader**

```
900 rem      print@ 4.2
902 rem  david priddle  august 27,1982
904 rem
906 rem
908 rem
910 rem '@' puts atn low,  ';' or '\' puts atn high
911 rem addresses converted to talk or listen as appropriate
912 rem print@(pa),(sa);a,b,c$...  sends data and cr only
914 rem print@(pa),(sa)\a$,b%,c..  sends data and cr/lf
916 rem input@(pa),(sa);a$,b... input ends on eoi,buffer full or cr
918 rem input@(pa),(sa)\a,b...  input ignores cr
919 rem
920 rem ************************************************************
921 rem
922 rem print!(pa),(sa);(bytel),(byte2),-(byte3)... minus sets eoi
924 rem                (byte-)=expression with value 0-255
925 rem print!;(bytel)...   allowed if device has been addressed
926 rem print!;      not allowed without either address or datum
927 rem addresses sent 'as is' - not converted to talk or listen
928 rem don't forget to unt/unl the buss
930 rem
932 rem input!a    input single byte and assign to variable
933 rem input!a$,b... input single bytes and assign to variables
934 rem device must be addressed with  'print!(pa),(sa);'  first
941 rem
942 rem (sa)  is optional in all cases if allowed by device
944 rem
1000 data169,126,133,49,133,53,169,70,133,48,133,52,169,76,133,112
1010 data169,229,133,113,169,126,133,114,96,32,112,0,162,0,32,192
1020 data241,157,124,2,232,36,16,48,4,201,13,240,10,224,250,240
1030 data6,165,150,41,64,240,231,169,34,157,124,2,232,169,58,157
1040 data124,2,32,209,127,32,118,0,208,207,76,223,126,169,0,133
1050 data150,32,209,200,134,212,134,176,134,175,173,123,2,240,7,138
1060 data32,210,240,24,144,4,138,32,213,240,32,118,0,201,59,240
1070 data14,201,92,240,10,32,209,200,134,211,134,165,32,9,241,32
1080 data72,241,32,118,0,201,92,208,4,169,255,133,16,173,123,2
1090 data208,135,32,112,0,32,170,186,32,180,187,76,250,126,140,122
1100 data2,230,119,208,2,230,120,160,0,177,119,201,153,240,10,201
1110 data133,240,9,172,122,2,76,118,0,169,0,44,169,34,141,123
1120 data2,32,112,0,201,64,240,133,201,33,240,6,32,247,127,76
1130 data250,126,173,123,2,208,94,32,112,0,201,59,240,46,32,247
1140 data127,32,209,200,134,212,134,165,32,255,240,32,118,0,201,59
1150 data240,10,32,209,200,134,211,134,165,32,9,241,32,72,241,32
1160 data112,0,240,175,32,247,127,32,118,0,240,167,32,112,0,201
1170 data171,240,14,32,247,127,32,209,200,134,165,32,9,241,76,78
1180 data127,169,52,141,17,232,32,209,200,134,165,32,9,241,169,60
1190 data141,17,232,16,210,169,34,141,125,2,169,58,141,126,2,173
1200 data64,232,41,253,141,64,232,32,112,0,32,192,241,141,124,2
1210 data133,165,32,118,0,208,29,169,34,141,125,2,169,58,141,126
1220 data2,169,8,13,33,232,141,33,232,169,2,13,64,232,141,64
1230 data232,76,250,126,32,209,127,32,118,0,240,219,32,112,0,32
1240 data192,241,141,124,2,133,165,76,187,127,32,43,193,133,70,132
1250 data71,165,119,72,165,120,72,36,7,48,3,169,124,44,169,123
1260 data133,119,169,2,133,120,32,60,185,104,133,120,104,133,119,96
1270 data165,119,208,2,198,120,198,119,96
1400 restore
1410 fori=32327to32767:readа:poke(i),a:next
1420 sys32327
1430 end
```

Samples of Command Use

```
PRINT@4,0;"string",a,b$
```

replaces

```
OPEN4,4,0:PRINT#4,"string",a,b$:
CLOSE4
```

```
INPUT@29,24;X
```

replaces

```
OPEN10,29,24:INPUT#10,X:CLOSE10
```

Idiosyncracies

While PRINT@3; will correctly print to the screen, INPUT@0; may *not* be used to input from the keyboard. This aspect of the input was not considered important for instrument control, but it could probably be changed if necessary.

PRINT!; requires that the user know the correct listen or talk address. These addresses may be formed by adding 32 or 64 to the device number. For example:

```
PRINT!36,0;65,66,13
PRINT!;67,68,13
```

will cause a printer device number 4 (listen address 36) to print A B (cr) and then C D (cr). ASCII values 65,66,67, 68,13 represent A,B,C,D,(cr). Note that the addresses are not sent in the second command since the device has not been unlistened and is still active on the bus. The UNLISTEN is sent using PRINT!63; .

When you use the INPUT! command, remember first to send a talk address using PRINT!(PA); and finally to use PRINT!95; to send the UNTALK.

Mr. Priddle may be contacted at the University of Toronto, Department of Chemistry, Toronto, Canada M5S 1A1.

MICRO

70 INCOME TAX PROGRAMS

(For Filing by April 15, 1983)

For APPLE II/II* (DOS 3.3, 16-Sector)

FEATURES:—

1. *Menu Driven.*
2. *70+ Tax Programs.*
3. *Basic; Unlocked; Listable.*
4. *Name/SS No./FS carried over.*
5. *Inputs can be checked.*
6. *Inputs can be changed.*
7. *I.R.S. approved REVPROC format.*
8. *Prints entire Form/Schedule.*
9. *Calculates Taxes, etc.*
10. *In 3.3 DOS, 16-Sector.*
11. *Fast calculations.*
12. *Use GREENBAR in triplicate — don't change paper all season!*
13. *Our 4th Year in Tax Programs.*
14. *We back up our Programs!*

Helpful programs to calculate and print the many Tax Forms and Schedules. Ideal for the Tax Preparer, C.P.A. and Individuals. For just $24.75 per disk, postpaid (in 3.3 DOS; 16-Sector disks).

Programs are designed for easy-use, with checkpoints to correct parts as needed. Results on screen for checking before printing.

In all, there are more than 70 individual Tax Programs. These include Form 1040, 1040A, 1040EZ, 1120, 1120S, 1041 and 1065. Also Schedules A, B, C, D, E, F, G, R, RP and SE. And, Forms 1116, 2106, 2119, 2210, 2440, 3468, 3903, 4255, 4562, 4797, 4835, 4972, 5695, 6251 and 6252.

And, we have a disk we call "THE TAX PREPARER'S HELPER" which has programs for INCOME STATEMENTS, RENTAL STATEMENTS, SUPPORTING STATEMENTS, IRA, ACRS, 1040/ES, ADD W-2's and PRINT W-2's.

TRY ONE DISK AND SEE FOR YOURSELF. ONLY $24.75 POSTPAID.

First disk is AP#1, and includes Form 1040 and Schedules A, B, C, D and G. $24.75 POSTPAID.

MasterCard

Write:—

GOOTH TAX PROGRAMS

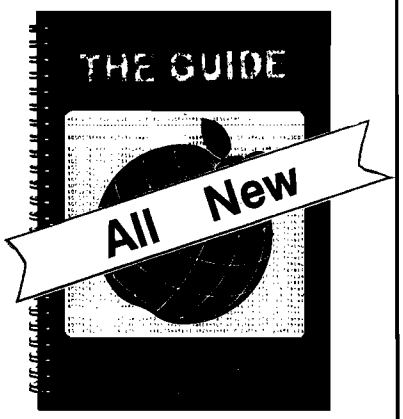
931 So. Bemiston • St. Louis, Mo. 63105

VISA

Announcing

THE GUIDE

A Complete Guide to the Apple Computer



If You Own the Original
What's Where in the APPLE?
You Will Want
THE GUIDE
only $9.95*

The Guide provides full explanatory text to lead you through the most complete Apple memory map ever published!

The Guide explains and demonstrates how to use the atlas and gazeteer published in the original volume!

MICRO makes it easy to order: Send check (payable to MICRO) to:

MICRO INK

P.O. Box 6502

Chelmsford, MA 01824

Call our **toll-free** number:

1-800-345-8112

(In PA, 1-800-662-2444)

VISA and MasterCard accepted

*Add $2.00 shipping per book.
MA residents add 5%.

COMPU SENSE

VIC-20®

VIC-20®	**Personal Computer**	$179.95
VIC-1011A	**RS232C Interface**	39.95
VIC-1515	**Printer**	334.95
VIC-1530	**Datasette**	67.50
VIC-1540	**Disk Drive**	349.95
VIC-1010	**Expansion Module**	139.95
VIC-1311	**Joystick**	9.95
VIC-1312	**Game Paddles**	19.95
VIC-1600	**Telephone Modem**	99.95
CM-151	**Terminal 40**	29.95

Produces 40 column output of information received through the modem

VIC-1210	**VIC 3K Memory Expander Cartridge**	34.95

Plugs directly into the VIC's expansion port. Expands to 8K RAM total.

VIC-1110	**VIC 8K Memory Expander Cartridge**	52.50

8K RAM expansion cartridge plugs directly into the VIC.

VIC-1011A	**RS232C Terminal Interface**	39.95

Provides interface between the VIC-20 and RS232 telecommunications modems. Connects to VIC's user port.

CM	**16K Memory Expander**	89.95
CM	**24K Memory Expander**	110.95

CARDBOARD 6 $87.95
An expansion interface for the VIC-20. Allows expansion to 40K or accepts up to six games. May be daisy chained for more versatility.

CARDBOARD 3 $29.95
Economy expansion interface for the VIC-20.

CARD "?" CARD/PRINT $79.95
Universal Centronics Parallel Printer Interface for the VIC-20 or CBM-64. Use an Epson MX-80 or OKIDATA or TANDY or just about any other.

CARDETTE $39.95
Use any standard cassette player/recorder with your VIC-20 or CBM-64.

CARDRITER $29.95
A light pen with six good programs to use with your VIC-20 or CBM-64

BUSINESS & HOME APPLICATIONS FOR VIC-20®

CW-107A	**Home Calculation Program Pack**	$48.95
CPV-31	**Data Files** - your storage is unlimited	14.95
CPV-96	**Household Finance Package** - to keep records of all your household expenses	30.95
CPV-208	**Bar-Chart** - display your numerical data	8.95
CH	**Turtle Graphics** - learn programming	34.95
CH	**VIC Forth** - is a powerful language for BASIC programming	49.95
CH	**HES MON** - is a 6502 machine language monitor with a mini-assembler	34.95
CH	**HES Writer** - time-saving word processing tool	34.95
CH	**Encoder** - keep your personal records away from prying eyes	34.95
CT-21	**Statistics Sadistics** - statistical analysis	14.95
CT-121	**Total Time Manager 2.0** - creates personal or business schedules	15.95
CT-124	**Totl Label** - a mailing list and label program	13.95
CT-125	**Totl Text BASIC**	15.95
CT-126	**Research Assistant** - keep track of reference data	17.50
CT-140	**Totl Text Enhanced**	29.95
CM-152	**Grafix Designer** - design graphic characters	12.95
CQ-5	**Minimon** - allows you to program, load, save, or execute machine language programs	13.95
CT-3	**Order Tracker**	15.95
CT-4	**Business Inventory** - to maintain record of inventory	15.95
CPV-210	**Bidder**	13.95
CPV-217	**Cash Flow Model** - determine cash flow	12.95
CPV-220	**Client Tickler**	16.95
CPV-221	**Club Lister**	13.95
CPV-224	**Depreciator**	9.95
CPV-236	**Investment Analyst** - keep track of investments and investment opportunitites	12.95
CPV-251	**Present Value**	10.95
CPV-269	**Super Broker**	12.95
CPV-270	**Syndicator** - calculates whether to buy or sell	13.95
CPV-274	**Ticker Tape** - maintains investments profile	14.95
CPV-276	**Un-Word Processor** - screen editor	16.95
CPV-286	**Phone Directory** - never lose a phone number again	9.95
CS-111	**Checkbook** - home "utility" program	14.95
CPV-294	**Calendar My Appointments** - print a calendar for every month in any year	14.95
CPV-296	**The Budgeter** - place your personal finances in order	12.95
CPV-327	**HESCOM** - transfers data and programs bidirectionally between VICs at three times the speed of a disk drive	40.95
CPV-328	**HESCOUNT** - monitors program execution	19.95
CHV	**HESPLOT** - Hi-res graphics subroutines	12.95
CPV-367	**Conversions** - figures, volume, length, weight, area, and velocity to all possible configurations	7.95
CC	**The Mail** - your complete mail program — Cassette	24.95
	Disk	29.95
CS	**Home Inventory** - lists your home belongings	17.95
CS	**Check Minder** - (V-20 & 64) keep your checkbook the right way	14.95
CS	**General Ledger** - a complete general ledger	19.95

BUSINESS & HOME APPLICATIONS FOR C-64

CHC-504	**HES Writer** - word processor	$39.95
CHC-503	**Turtle Graphics II** - utilizes the full graphics of your 64	49.95
CHC-502	**HESMON** - machine language monitor w/mini-assembler	34.95
CHP-102	**6502 Professional Development System**	29.95
CFC	**Data Files** - a management program	27.95

MANY MORE PROGRAMS FOR YOUR 64 & 20

COMMODORE SOFTWARE

VIC-1211A	**VIC-20 Super Expander**	$57.99

Everything Commodore could pack into one cartridge - 3K RAM memory expansion, high resolution graphics plotting, color, paint and sound commands. Graphic, text, multicolor and music modes. 1024x1024 dot screen plotting. All commands may be typed as new BASIC commands or accessed by hitting one of the VIC's special function keys. Includes tutorial instruction book. Excellent for all programming levels.

VIC-1212	**Programmer's Aid Cartridge**	$45.99

More than 20 new BASIC commands help new and experienced programmers renumber, trace and edit BASIC programs. Trace any program line-by-line as it executes, pause to edit. Special KEY command lets programmers redefine function keys as BASIC commands, subroutines or new commands.

VIC-1213	**VICMON Machine Language Monitor**	$48.99

Helps machine code programmers write fast efficient 6502 assembly language programs. Includes one line assembler/disassembler.

GAMES FOR YOUR VIC-20®

CCS	**Cribbage**	$14.95
CCD	**Motor Mouse**	12.99
CW-1901	**Avenger Cart.** - an invasion of space intruders and you're the VIC "Avenger"	24.95
CW-1904	**Superslot Cart.** - great music and sound effects!	24.95
CW-1906	**Super Alien Cart.** - you're trapped in a maze	24.95
CW-1907	**Jupiter Lander Cart.** - pilot your "Jupiter Lander"	24.95
CW-1908	**Draw Poker Cart.**	24.95
CW-1909	**Midnight Drive Cart.** - authentic night driving	24.95
CW-1910	**Radar Rat Race**	24.95
CW-1911	**Sky Falling**	24.95
CW-1912	**Mole Attack** - a colorful "cartoon action" game	24.95
CW-1913	**Raid on Ft. Knox** - try to escape the guards	24.95
CW-1914	**Adventure Land** - Formerly available only on larger, more expensive computers. All Adventure games are decoded to "talk" on the Type N Talk voice synthesizer (available from VOTRAX)	31.95
CW-1915	**Pirate Cove Adventure** - Yo, ho, ho, & a bottle of rum	31.95
CW-1916	**Mission Impossible Adventure**	31.95
CW-1917	**The Count Adventure** - trapped in Dracula's castle with 3 days to find and destroy the vampire	31.95
CW-1918	**Voodoo Castle Adventure** - you have to free Count Yorga from a curse	31.95
CW-1919	**Sargon II Chess** - seven challenging play levels	31.95
CW-1923	**Gorf** - (The smash-hit arcade game!)	31.95
CW-1924	**Omega Race** - the ultimate space game	31.95
CW-1937	**Seawolf** — an explosive Bally Midway arcade "classic"	24.95
CH-G202	**Maze of Mikor** - adventure-packed game with stunning graphics	15.95
CH-G203	**Tank Wars**	15.95
CH-G205	**Pinball**	13.45
CH-G206	**Simon** - It gets tougher as you get better. Great for kids of all ages	13.45
CH-G207	**Fuel Pirates**	13.45
CH-G209	**Laser Blitz**	15.95
CH-G210	**Tank Trap**	15.95
CH-G211	**Concentration**	13.45
CH-G212	**Dam Bomber** - pilot your plane, avoid enemy fire	13.45
CH-C307	**Shamus** - search room after room for the shadow-eluding androids, two levels of intense arcade action	34.95
CH-C308	**Protector**	36.95
CPU-79	**Breakout**	7.95
CPU-85	***Hangman*** - *unbelievable graphics and sound*	9.95
CPU-87	**Memory** - VIC challenges your memory	9.95
CPU-88	**Match** - hand and eye coordination	7.95
CPU-89	**Monks** - a devilish game of logic	7.95
CPU-108	**Bomber** - you must decide who you want to fly for, then pick a target and your experience level	9.95
CPU-109	**Amok** - the halls of Amok are populated by robots that obey one instruction - get the intruder	20.95
CPU-153	**Tank vs. UFO** - the tank is moving back and forth along the base, shoot the UFO before it shoots you	9.95
CPU-194	**Snakman** - Pacman for the VIC	14.95
	Defender on Tri - you're the pilot of the experimental ship, Defender	17.95
	3-D Man - the popular arcade game, requires 3K	17.95
	Exterminator - a game full of bugs	20.95

GAMES FOR YOUR 64

CCS	***Cribbage***		$17.95
CFC	**Flight 64** - what a program!	Cassette	14.95
		Disk	16.95
CFC	**Spright Generator**	Cassette	15.95
		Disk	17.95
	Mastermind		19.95
	Star Trek		9.95
	Black Jack		11.95
	Tic-Tac-Toe		7.95
	Backgammon		14.95
	Maze 64		15.95

Prices subject to change.

TO ORDER:
P.O. Box 18765
Wichita, KS 67218
(316) 684-4660

MasterCard VISA

Personal checks accepted (Allow 3 weeks)
or C.O.D. (Add $2) *Handling charges $2.00*

VIC-20® is a registered trademark of Commodore

VIC RS-232 Printer

by Michael V. Tulloch

The hardware and software needed to interface RS-232 devices to Commodore's VIC computer are described. A Radio Shack Line Printer VII is used as an example. Other examples include hardware to convert RS-232 voltages to TTL, a BASIC program to convert hex code data to POKEs, a machine-language printer driver, and a discussion on using a CTS line from VIC.

Printer Driver
requires:

VIC-20
RS-232 printer
hardware interface

The Programmers Reference Guide (PRG) is indispensable for understanding the VIC's RS-232 implementation. Unfortunately, some of the information is misleading.

Part of PRG's chapter four is devoted to the RS-232 interface. Although quite versatile, VIC software does not implement two RS-232 functions that may be important with many printers. Neither "ring indicator" (RI) nor "clear to send" (CTS) is included.

Unless a printer has a large buffer to allow simultaneous data input while printing, some handshaking is required. The simplest approach involves waiting long enough after each output to the printer to be sure the printer has received, printed, and returned the print head. In BASIC this is easily done; from the command mode it is not.

Standard printer handshaking involves one line. While it is busy, the printer sends out a signal on this line. The computer reads this busy line and holds off further output until the printer is free. A busy line usually connects to RS-232 CTS — not used by VIC's software.

After referring to the VIC PRG and considerable experimentation, I determined the only way to use the CTS line was to write a program to read it directly.

Hardware

Another problem, caused by VIC's non-standard version of RS-232, involves voltage levels. VIC outputs 0 to +5 volts — TTL levels. VIC also expects 0 to +5 volts for any inputs. You must observe these limitations since the RS-232 lines connect directly to a 6522 VIA. Because RS-232 standard devices expect to be sent ±12 volts and usually output the same voltage range, there is an obvious incompatibility.

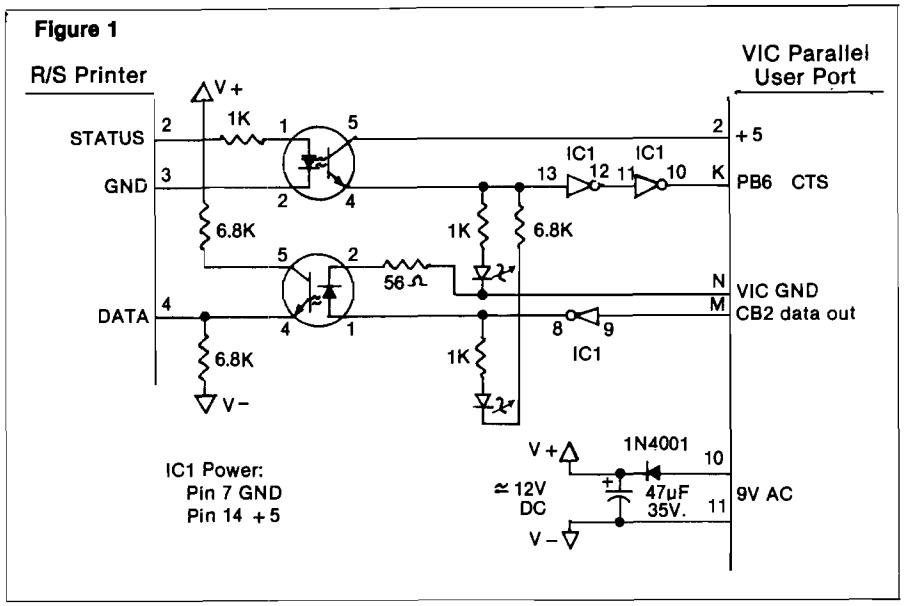
The circuit shown in figure 1 adjusts voltage levels. It converts VIC's 5-volt signal to ±12 volts (actually ±10 volts — OK for most applications) and *vice versa*. Although, there are several ways to accomplish this conversion, the approach chosen is capable of isolating the VIC completely from the printer if the printer supplies the ±12 volts or a separate ±12-volt supply is used.

A 74LS04 hex inverter protects the input and output lines of the RS-232 port. This inverter also corrects VIC's signal inversion problem. An opto-isolator shifts the level from 0-5 volts to ±12 volts.

In this example a single rectifier and electrolytic capacitor converts VIC's ±9-volt ac output to about ±12 volts dc. A resistor divider provides a ground for the printer. Since RS-232 inputs draw very little current, this circuit works adequately. Note that the protective ground and signal ground of the VIC are at the same potential and are not isolated. Do not connect either to the printer ground.

The pin connections shown in figure 1 are for the Radio Shack printer. Note that Radio Shack calls the busy line the 'STATUS' line. Any three-wire cable will do. if you have an RFI problem (and to comply with the new rules

Figure 1



for Part B computers) use a shielded cable. Connect the shield either to the printer ground or VIC's protective ground (*not both!*).

As an aid to trouble shooting, I included a couple of LEDs. Although they increase the current required by the circuit, they may be helpful in diagnosing interfacing problems. Besides, I enjoy watching them blink as VIC talks on the RS-232 port.

Although rather primitive, the interface can be constructed out of junk box parts. It's so cheap that even if you buy all new parts in five packs at Radio Shack, the bill will come to only $12.00 (see table 1).

Table 1: Parts List

Cost	# Used	Item	Radio Shack Part #
$ 1.98	2	Opto-isolators	276-1628*
0.79	1	74LS04 Hex Inverter	276-1904
1.19	2	LEDs	276-032
0.39	4	6.8K Resistor ¼W	271-1333
0.39	3	1K Resistor ¼W	271-1321
0.10	1	56 OHM Resistor	(Assortment)
0.49	1	1N4001 Diode	276-1101
0.69	1	47 F 35V Electrolytic	272-1015 or 272-1026
2.99	1	Edge Connector (Cut to 24 pins)	276-150 or 276-151
1.49	1	4-Pin DIN Plug	Available some stores
1.29	1	14-Pin DIP Socket	276-1993
		3-Wire and 6-Wire Cable	

$11.79 (plus tax) total

*This part has been discontinued by Radio Shack and may not be available in all stores. Nearly any opto-isolator will do.

VIC uses a 6522 VIA (versatile interface adapter) for the RS-232 port. The same chip also drives the user port. Although VIC has two VIAs, we're interested in chip #1. Memory addresses $9119-$911F (37136-37151) access this chip. The RS-232 software sets all the control registers. Therefore we won't have to do it and I'll ignore the process. As I've mentioned, however, the CTS line is not read by the RS-232 software. This line is tied to PB6 (pin K). PB6 is one of eight data lines of the B VIA port I/O register. The B port register can be read or written to at memory address $9110 (37136).

Normally all VIA pins are either defined as outputs or pulled up to 5 volts. PEEKing the port register should yield a value of 255. In other words, all pins (outputs and +5 volts) are seen as logical 1. If pin K is connected to the printer's 'busy' line so that the busy condition pulls the line to 0 volts, then PEEKing $9110 will return a value of 191 when the printer is busy. To recognize a busy printer you could read the B port and look for a value of 191.

Printer Driver Software

As Murphy would have it, things

EVER WONDER HOW YOUR APPLE II WORKS?

***QUICKTRACE* will show you! And it can show you WHY when it doesn't!**

This relocatable program traces and displays the actual machine operations, *while* it is running *and* without interfering with those operations. Look at these *FEATURES:*

Single-Step *mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

Trace *mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

Background *mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

QUICKTRACE *allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

Two optional display formats *can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

QUICKTRACE *is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

QUICKTRACE *is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

QUICKTRACE *is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while* ***QUICKTRACE*** *is alive.*

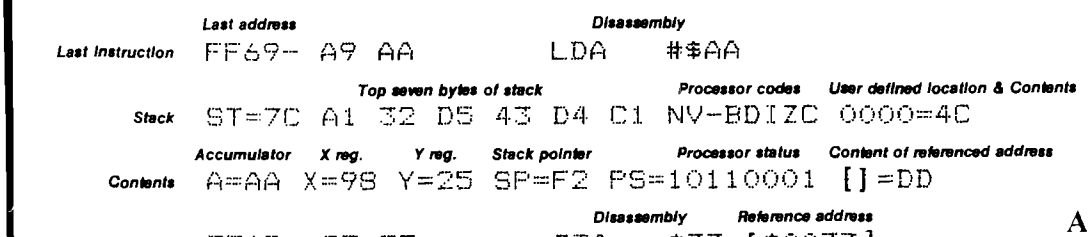
QUICKTRACE *is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

Price: $50

QUICKTRACE requires 3548 ($E00) bytes (14 pages) of memory and some knowledge of machine language programming. It will run on any Apple II or Apple II Plus computer and can be loaded from disk or tape. It is supplied on disk with DOS 3.3.

QUICKTRACE was written by John Rogers.
QUICKTRACE is a trademark of Anthro-Digital, Inc.

QUICKTRACE DEBUGGER



Anthro-Digital, Inc.
P.O. Box 1385
Pittsfield, MA 01202
413-448-8278

aren't entirely that simple. VIC, both too smart and too dumb, has a 512-byte RS-232 buffer. When printing to the RS-232 port, characters go first to the buffer. If you don't print more than 512 characters, the program will continue running without waiting for the printer. The RS-232 software sends characters out simultaneously with other VIC functions (except serial port and cassette).

The "too dumb" part involves VIC's failure to use the CTS line. Instead you have to handle the busy line yourself. My solution to this problem gives up the advantages of the printing buffer. The machine-language program described here only allows characters to be output to the buffer at print speed. It holds up printing to the buffer while the printer returns its print head. This slows down RS-232 communications considerably, but it works.

To intercept the characters as they are printed to the buffer, the output vector at $0326-$0327 (806-807) must be changed. For the routine used here they must be changed to point to the cassette buffer $033C (828) where our machine-language program lives. This vector can be changed either from BASIC or the keyboard.

POKE 806,60 lo byte
POKE 807,03 hi byte

Once you have built the interface and an appropriate cable, the next step is to interface the software — not a simple task. Since I don't have a VIC machine-language monitor, I used an assembler on my Apple. Listing 1 is the resulting assembly code. Liberal comments are provided, so I'll only briefly describe the routine. Two delay loops are used. The first loop checks the CTS bit until $8F successive checks show it has gone high, which means the printer is not busy. The second delay executes if the last character printed was a carriage return. These two delays assure that spaces between characters are not misinterpreted and that the routine sees the print head return to home.

Because the listed version uses four locations at the top of the VIC's screen, you can watch the characters pass to the printer and the various timing loops perform. The comment section at the top of the listing suggests alternative storage locations for permanent use. Don't worry about the screen scrolling — the routine is not affected.

Listing 1

```
                        1   ********************************
                        2   *                              *
                        3   *             VIC              *
                        4   * OUTPUT ROUTINE FOR RS-232     *
                        5   *V2.2                           *
                        6   *                              *
                        7   *             BY               *
                        8   *      MICHAEL TULLOCH
                        9   *         17 AUG 82             *
                       10   *                              *
                       11   * THIS ROUTINE USES THE CTS     *
                       12   * LINE WHICH IS NOT IMPLIMENTED*
                       13   * BY THE VIC SOFTWARE.         *
                       14   *                              *
                       15   *                              *
                       16   *             NOTES:           *
                       17   *                              *
                       18   *  1. DIFFERENT PRINTERS MAY   *
                       19   *     REQUIRE DIFFERENT CNTR    *
                       20   *     VALUES.                  *
                       21   *  2. DIFFERENT DELAY VALUES   *
                       22   *     MAY BE NEEDED.           *
                       23   *  3. LOCATIONS OF SAVA,SAVX,  *
                       24   *     FLAG, AND DELAY COUNTER  *
                       25   *     MAY BE CHANGED TO -
                       26   *          SAVA EQU $0380      *
                       27   *          SAVX EQU $0381      *
                       28   *          FLAG EQU $0382      *
                       29   *         DCNTR EQU $0383      *
                       30   *                              *
                       31   *                              *
                       32   ********************************
                       33
                       34            ORG  $033C
                       35   *
                       36   *  THIS IS THE START OF THE CASSETTE BUFFER
                       37   *
                       38   SAVA     EQU  $1E00      SAVE ACCUMULATOR
                       39   FLAG     EQU  $1E02      FLAG FOR C/R AS LAST CHARACTER
                       40   OUT      EQU  $F27A      REAL CHARACTER OUTPUT
                       41   SAVX     EQU  $1E01      SAVE X REGISTER
                       42   CNTR     EQU  $8F        EMPIRICAL DELAY BETWEEN SPACES
                       43   DCNTR    EQU  $1E01
                       44   BPORT    EQU  $9110      LOCATION OF RS232 CTS REGISTER
033C: 8D 00 1E         46            STA  SAVA       SAVE ACCUMULATOR
033F: 8E 01 1E         47            STX  SAVX       SAVE X REGISTER
0342: AD 02 1E         48            LDA  FLAG       GET THE FLAG
0345: C9 0D            49            CMP  #13        WAS IT CR
0347: F0 24            50            BEQ  DELAY      IF SO THEN WAIT A WHILE
0349: A9 00            51            LDA  #0         NO SO RESET FLAG
034B: 8D 02 1E         52            STA  FLAG       TO ZERO
034E: A6 8F            53   RESET    LDX  CNTR       SET UP NUMBER OF TIMES TO LOOK
0350: AD 10 91         54   LOOK     LDA  BPORT      GET THE CTS SIGNAL
0353: C9 BF            55            CMP  #191       IS IT LOW?
0355: F0 F7            56            BEQ  RESET      YES SO START LOOKING ALL OVER
0357: CA               57            DEX  NO         SO DECRIMENT NUMBER OF LOOKS
0358: D0 F6            58            BNE  LOOK       TAKE ANOTHER LOOK
                       59   * FINALLY, ALL LOOKS GOOD
035A: AD 00 1E         60            LDA  SAVA       SO, RESTORE ACCUMULATOR
035D: AE 01 1E         61            LDX  SAVX       RESTORE X REGISTER
0360: C9 0D            62            CMP  #13        IS OUTPUT VALUE A C/R?
0362: F0 03            63            BEQ  FLAGSET    YES, SO SET FLAG
0364: 4C 7A F2         64            JMP  OUT        NO SO OUTPUT THE CHARACTER
                       65   *                        DON'T COME BACK
0367: 8D 02 1E         66   FLAGSET  STA  FLAG       SET C/R FLAG TO 13 (=$0D)
036A: 4C 7A F2         67            JMP  OUT        SET, SO OUTPUT THE CHARACTER,
                       68   *                        DON'T COME BACK
036D: A9 FF            69   DELAY    LDA  #$FF       MAKE IT A LONG DELAY
036F: 8D 01 1E         70            STA  DCNTR      SAVE ACCUMULATOR AT TOP OF SCREEN
0372: A2 FF            71   XSTRT    LDX  #$FF       MAKE IT A REALLY LONG DELAY
0374: CA               72   XDEC     DEX  START      COUNTING OUT
0375: D0 FD            73            BNE  XDEC       IF X IS NOT ZERO THEN
                       74   *                        DECRIMENT SOME MORE
0377: CE 01 1E         75            DEC  DCNTR      DECRIMENT THE OUTSIDE COUNTER
037A: D0 F6            76            BNE  XSTRT      IF NOT 0 DO INNER LOOP AGAIN
037C: 4C 49 03         77            JMP  $0349      DONE AT LAST RETURN FROM DELAY
```

To make entering the machine-language program easier I wrote the BASIC program in listing 2. Note that the data statements contain the code in hex. The program reads the hex data and converts it into decimal numbers. It also reads the start address from a data statement, then POKEs the data into successive memory locations. By removing lines 150 and 160 and changing the data in lines 5-17, this program will convert and enter any hex code sequence. For those of you who don't have VICMon, this is easier than converting hex to decimal and typing it in as data or typing a lot of POKEs.

Using the programs is easy; just follow these instructions:

1. Enter the BASIC program.
2. SAVE the BASIC program.
3. RUN the BASIC program.
4. Either from the keyboard or from within a program,

   OPEN2,2,3,CHR$(7) + CHR$(0)

5. To LIST to an RS-232 printer
   CMD 2

Listing 2

```
1 GOTO 100
5 DATA 828: REM START OF CASETTE BUFFER
8 REM.
9 REM MACHINE LANGUAGE PROGRAM AS HEX DATA
10 DATA 8D,00,1E,8E,01,1E,AD,02,1E,C9,0D,F0,24,A9,00
11 DATA 8D,02,1E,A6,BF,AD,10,91,C9,BF,F0,F7,CA,D0,F6
12 DATA AD,00,1E,AE,01,1E,C9,0D,F0,03,4C,7A,F2,8D,02,1E,4C,7A,F2,A9,FF,8D,04,1E
13 DATA A2,FF,CA,D0,FD,CE,04,1E,D0,F6,4C,49,03
17 DATA END
18 REM ***** END DATA *****
19 REM       HEX TO DECIMAL SUBROUTINE
20 READ A$:IF A$="END" THEN 98
25 IF LEN(A$)=1THEN 40:REM JUST ONE BIT SO SKIP HI BIT STUFF
30 A1$=LEFT$(A$,1):GOSUB 80:A=VAL(A1$):REM DO HI BIT
40 A1$=RIGHT$(A$,1):GOSUB 80:B=VAL(A1$):REM DO LO BIT
50 V=(A*16)+B:REM MAKE IT BASE TEN
70 A=0:B=0:REM KEP IT CLEAN
75 RETURN
79 REM     CHANGE LETTERS TO NUMBERS SUBROUTINE
80 IF VAL(A1$)>0 THEN RETURN:REM IT IS A NUMBER
82 IF A1$="0"  THEN RETURN : REM IT IS A ZERO
83 REM IT WASN'T A NUMBER SO IT MUST BE A LETTER
84 Z=ASC(A1$): REM CHANGE TO A NUMBER
86 A1$=STR$(Z-55):REM MAKE NUMBER 10 TO 15
90 REM A NEAT TRICK TO CONVERT A-F TO 10-15
98 RETURN
99 REM *************   MAIN  *************
100 REM POKE  THE MACHINE LANGUAGE
110 READ S:REM GET THE STARTING MEMORY LOCATION
120 GOSUB 20: REM CHANGE HEX TO BASE TEN
130 POKE S,V:REM DO IT
135 PRINTS,V:REM SO WE CAN SEE IT
137 S=S+1:REM INCRIMENT THE MEMORY LOCATION
138 V=0
140 IF A$<>"END" THEN 120
150 POKE36879,8:REM MAKE SCREEN BLACK SO WE CAN SEE IT WORK
160 POKE 806,60:POKE807,03:REM SET THE OUT PUT VECTOR TO ROUTINE
190 END
```

LIST
PRINT#2
CLOSE 2
RUN/STOP - RESTORE

Remember to PRINT# before CLOSEing the RS-232 channel. If the busy line goes low and stays there, the system will hang. If you unsuccessfully try to run/stop-restore, you may have a printer problem. Good luck and happy PRINT#ing!

References

1. Butterfield, J. and Law, Jim, *Compute!*, Vol. 4, No. 8, August 1982, p. 99.
2. Finkel, A., et. al., *Programmer's Reference Guide*, Commodore Business Machines, 1982.
3. Lesea, A. and Zaks, R., *Microprocessor Interfacing Techniques*, Sybex, Berkeley, CA, 1977.

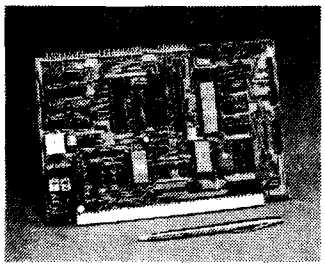
Michael Tulloch is a consultant in engineering psychology and president of Intelligent Home Systems, Inc. He may be contacted at Intelligent Home Systems, Inc., P.O. Box 0858, Roswell, GA 30077.

MICRO

QCB-9 SINGLE BOARD COMPUTER

- **6809 BASED**
- **RUNS TSC FLEX DOS**
  - ★ QCB-9/1 S-100 BUS
  - ★ QCB-9/2 SS-50 BUS

$149.00* *PARTIAL KIT



FEATURES

- 5¼" Floppy Controller
- Serial RS-232 Port
- Centronics Type Printer Port
- Keyboard/Parallel Port
- 24K Bytes of Memory
- QBUG Resident Monitor
- 6802 Adaptor

FULLY ASSEMBLED & TESTED $389.00

- 48-hour Burn-in
- 90 Day Warranty

NAKED-09 SS-50 6809 CPU CARD $49.95*
★ 1K OF RAM AT E400
★ 6K OF EPROM AT E800-FFFF
Assembled & Tested $149.00
2 MHZ Version $189.00
*PCB & Documentation Only
★ HIGH QUALITY DOUBLE SIDED PCB ★ SOLDER MASKED ★ SILK SCREENED

TSC, FLEX DOS, ASSEMBLER, EDITOR $150.00

QBUG RESIDENT MONITOR $50.00
★ Disc Boot ★ Memory Test ★ Break Points
★ Memory Exam & Exchange ★ Zero Memory ★ Jump to User Program
★ Memory Dump ★ Fill Memory ★ Register Display & Change
QBUG IS A TRADEMARK OF LOGICAL DEVICES INC., ® Copyright 1981

PHONE ORDERS: (305) 776-5870

LOGICAL DEVICES INC.
COMPUTER PRODUCTS DIVISION
781 W. OAKLAND PARK BLVD. • FT. LAUDERDALE, FL 33311
TWX: 510-955-9496 • WE ACCEPT VISA, MC, CHECKS, C.O.D., MONEY ORDER

PREMIER ISSUE
DECEMBER 1, 1982
Commander

THE MONTHLY JOURNAL FOR COMMODORE COMPUTER USERS
VIC - 20 64 PET/CBM
MAX MACHINE

"COMMANDER will be dedicated to communicating the fun of, as well as the latest information about the COMMODORE COMPUTERS."

GET YOUR MONEY'S WORTH

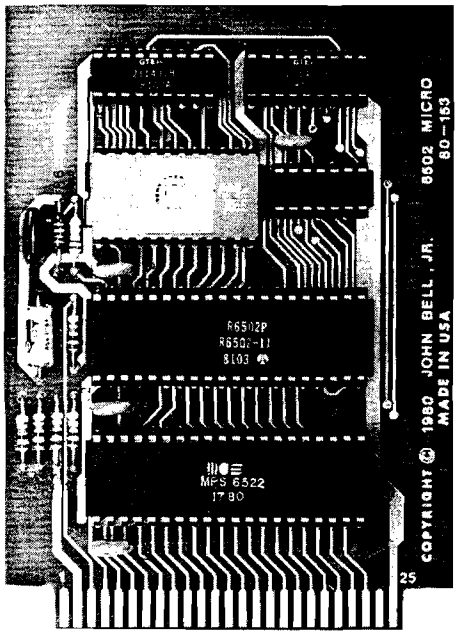
You've probably made a sizeable investment in your computer equipment. COMMANDER can help you make the most of it. Each issue brings you the *no-nonsense* advice you need to stay on the leading edge of this constantly changing field. COMMANDER will be your reference source to the world of computers... with the best, most comprehensive coverage you can get!!

PREMIER ISSUE ☐ 1 YR. $22 ☐ 2 YR. $40 ☐ 3 YR. $56
$4 DISCOUNT *(PRICES DO NOT INCLUDE $4 DISCOUNT)*

— Subscription Orders Only —
Toll Free Number: 1-800-426-1830
(except WA, HI, AK)

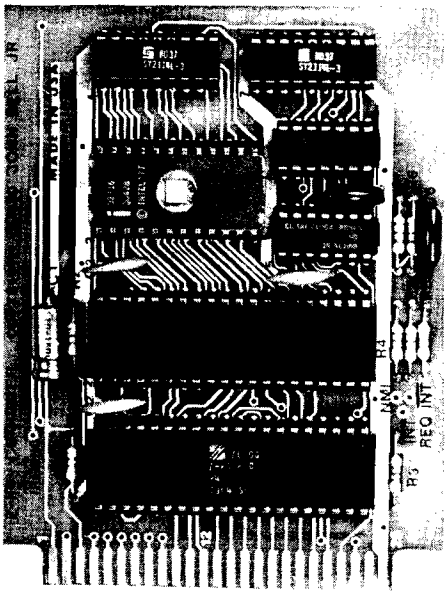
COMMANDER
P.O. BOX 98827
TACOMA, WASHINGTON 98498
(206) 565-6818

INDUSTRIAL CONTROL MICROCOMPUTERS



6502 AND Z80 MICROCOMPUTERS ARE DEDICATED COMPUTERS DESIGNED FOR **CONTROL FUNCTIONS.**

THESE BOARDS FEATURE:
- 4096 BYTES EPROM
- 1024 BYTES RAM
- ALL BOARDS INCLUDE COMPLETE DOCUMENTATION
- 50 PIN CONNECTOR INCLUDED
- EPROM AVAILABLE SEPARATELY



JOHN BELL ENGINEERING'S 6502 MICROCOMPUTER FEATURES:

- 1024 BYTES RAM
- 4096 BYTES EPROM
- USES ONE 6522 VIA (DOC. INCL.)
  - 2 8 BIT BIDIRECTIONAL I/O PORTS
  - 2 16 BIT PROGRAMMABLE TIMER/ COUNTERS
  - SERIAL DATA PORT
  - LATCHED I/O WITH HANDSHAKING LOGIC
- TTL AND CMOS COMPATIBLE

80-153A	LIST	100-499
EPROM NOT INCLUDED	$110.95	$66.57

JOHN BELL ENGINEERING'S NEW Z80 MICROCOMPUTER FEATURES:

- Z80 CPU-SOFTWARE COMPATIBLE WITH Z80, 8080 AND 8085 MICROPROCESSORS
- 4096 BYTES EPROM
- 1024 BYTES RAM
- SINGLE 5V POWER SUPPLY AT 300MA
- CLOCK FREQUENCY IS 2MHZ, RC CONTROLLED
- Z80 PIO (DOC. INCL.)
- 2 8-BIT BIDIRECTIONAL I/O PORTS
- LATCHED I/O WITH HANDSHAKING LOGIC
- TTL AND CMOS COMPATIBLE

80-280A	LIST	100-499
EPROM NOT INCLUDED	$129.95	$77.97

USE YOUR 6502 OR Z80 MICROCOMPUTER TO CONTROL EVERYTHING!

- YOUR HOME SECURITY SYSTEM
- HEAT CONTROL
- LIGHT CONTROL
- SOLAR HEATING AND POWER SYSTEMS
- AUTOMATIC CONTROL OF TAPE RECORDERS
- TRAFFIC LIGHT CONTROL
- IRRIGATION SYSTEMS
- AUTOMATIC CONTROL OF VIDEO RECORDERS
- ROBOT CONTROL
- AUTOMATIC DIALER
- AUTOMATED SLIDE SHOW CONTROL
- COMMUNICATION SYSTEMS FOR THE DISABLED
- THE WORLD

JOHN BELL ENGINEERING, INC.

ALL PRODUCTS ARE AVAILABLE FROM **JOHN BELL ENGINEERING, INC.** • 1014 CENTER ST., SAN CARLOS, CA 94070
ADD SALES TAX IN CALIFORNIA • ADD 5% SHIPPING & HANDLING 3% FOR ORDERS OVER $100

MasterCard

SEND $1.00 FOR CATALOG

(415) 592-8411
WILL CALL HOURS: 9am-4pm

10% OUTSIDE U.S.A.
ADD $1.50 FOR C.O.D.

VISA

#300

PROM BASIC for the C1P

by David A. Jones

This article describes a unique way to increase the performance of the Superboard II/C1P without a disk interface and drive. The modifications are for the user who wishes to dedicate his system to only a few specific tasks that are run often, or who is inexperienced in the use of sophisticated computers.

PROM BASIC
requires:
C1P

As an alternative to upgrading a cassette-based system to disk where improved performance is attained at a considerable price, I would like to share my ideas on a PROM BASIC system. Even with the general decline of computer hardware prices, adding a disk to my C1P would cost twice the original investment. The alternative I refer to is EPROMs. 2732 EPROMs are available for approximately $10 apiece, so for less than $100 you can buy eight EPROMs, a 24-pin ZIF connector, a couple of 74LS139 decoders, and still have enough left over to pay the sales tax.

There have been several articles in MICRO (39:97 and 45:31, for example) to expand the 600 board *via* the J1 connector. I won't go into detail here except to say I limited my memory expansion to 24-pin devices. The byte-wide RAM parts are now more cost effective than 2114's when you consider the price of sockets and power dissipation, and they have the added advantage of being pin-compatible with 2716 and 2732 EPROMs.

An expansion board with eight 24-pin sockets will utilize all of the generally available unused memory in the 600 board if populated with 2732 EPROMs, or half of that amount if

Listing 1

```
 10 0000                ;AUTOBASIC,11-2-81
 20 0000                ;BY DAVID A. JONES
 30 0000                ;
 40 97A0                *=$97A0
 50 97A0                VTMP  =$FA
 60 97A0                ADDR  =$FC
 70 97A0                YTMP  =$FE
 80 97A0                XTMP  =$FF
 90 97A0                IVEC  =$0218
100 97A0                WARM  =$A274
110 97A0                LEGAL =$FE93
120 97A0                ROLA  =$FEDA
130 97A0                INPUT =$FEED
140 97A0                BASOUT=$FFEE
150 97A0                ;
160 97A0 A92A           LDA #'*     PROMPT
170 97A2 20EEFF         JSR BASOUT
180 97A5 A204           LDX #4
190 97A7 20EDFE  INNN   JSR INPUT   GET ADDRESS (4 HEX DIGITS)
200 97AA 20EEFF         JSR BASOUT
210 97AD 2093FE         JSR LEGAL   HEX CHARACTER ?
220 97B0 86FF           STX XTMP
230 97B2 A200           LDX #0
240 97B4 20DAFE         JSR ROLA    ROLL IT INTO ADDRESS REGISTER
250 97B7 A6FF           LDX XTMP
260 97B9 CA             DEX
270 97BA D0EB           BNE INNN    GET REST OF ADDRESS
280 97BC                ;
290 97BC AD1802         LDA IVEC    SAVE INPUT VECTOR
300 97BF 85FA           STA VTMP
310 97C1 AD1902         LDA IVEC+1
320 97C4 85FB           STA VTMP+1
330 97C6                ;
340 97C6 ADDA97         LDA MOVE+1  SET INPUT VECTOR
350 97C9 8D1802         STA IVEC
360 97CC ADDB97         LDA MOVE+2
370 97CF 8D1902         STA IVEC+1
380 97D2 A000           LDY #0
390 97D4 84FE           STY YTMP    PRESET Y REGISTER
400 97D6                ;
410 97D6 4C74A2         JMP WARM
420 97D9                ;
430 97D9                ;
440 97D9 4CDC97  MOVE   JMP LOAD
450 97DC A4FE    LOAD   LDY YTMP    LOAD PROGRAM FROM PROM
460 97DE B1FC           LDA (ADDR),Y
470 97E0 C999           CMP #$99    END OF FILE ?
480 97E2 F008           BEQ OUT
490 97E4 C8             INY
500 97E5 84FE           STY YTMP
510 97E7 D002           BNE RETN
520 97E9 E6FD           INC ADDR+1  NEXT BLOCK
530 97EB 60      RETN   RTS
540 97EC                ;
550 97EC A5FA    OUT    LDA VTMP    RESET VECTOR
560 97EE 8D1802         STA IVEC
570 97F1 A5FB           LDA VTMP+1
580 97F3 8D1902         STA IVEC+1
590 97F6                ;
600 97F6 4C74A2         JMP WARM
610 97F9                ;
```

(continued)

HM6116 or TMM2016 RAMs are used. I chose a mixture of 4K RAM and 24K EPROM, which works well for a cassette-based system.

I originally built my expansion interface to house the Assembler/Editor and Extended Monitor in EPROM so they always would be available immediately. I wired in extra sockets so I could do the same with some assembly-language games for my children. I figured if video games have programs in cartridge packs so can my C1P. The Assembler and Extended Monitor are now as accessible as ROM BASIC. The Extended Monitor can be run directly from PROM, but since the Assembler/Editor has self-modifying code it must be moved to RAM to run. I wrote a short routine to do this automatically when I call the program. The response is instantaneous.

The next logical question was, if I can do it with machine-language programs, why not BASIC programs also? Some obvious answers are that BASIC programs are stored in token form, they tend to occupy more memory, they depend upon pointers in page zero that are set when the program is loaded, and these pointers are modified as the program runs. None of these reasons seemed insurmountable, although they were not without challenge.

I tried storing page zero along with the tokenized program and then loading both into RAM when I wanted the program. A problem occurred since my storing and loading routines also used page zero and therefore modified what I wanted to save. Also, storing page zero required 256 more bytes of storage and more complicated code to handle it unless page two and three were stored, which used still more space.

This method worked, but I decided to try another tack — emulating a load from the serial port. The advantage of this procedure is that you don't have to use page zero and the application program can be stored in ASCII rather than token form. Two or more programs can be chained, or program and data can be loaded independently. My word processor is in BASIC so I must retain this capability. The disadvantage is that ASCII takes more storage space. However, for moderate sized programs I find not storing page zero makes the tradeoff acceptable.

To load a program from EPROM, a three-part machine-language routine (listing 1) is called from BASIC *via* the USR(X) function. POKE 11,160:POKE 12,151:X=USR(X) for the addresses in listing 1. Once called, this routine changes the input vector to point to the second part of the routine. The second part feeds data to the BASIC load routine just as if it were coming from the serial port or keyboard. A check is made to see if the data contains a $99, which indicates the end of the load and, if so, the input vectors are reset to their original value by part 3. Then it jumps back to BASIC warm start. This routine allows the stored program to be located anywhere in memory and prompts the user with an asterisk (*). The user responds with the starting address.

Storing the program in EPROM is more complicated but by no means difficult once you understand the concept. Also, you must have access to an EPROM programmer.

First, be sure the program to be stored is thoroughly debugged and user friendly. Remove REM statements and, if you want, pack code to save memory. Save the resulting code on cassette tape and load the program back into memory with the routine in listing 2. The file is now stored in a straight ASCII format with no tokens. Find the end of the file, delete the OK, and insert a $99.

The last available RAM location is stored by BASIC cold start in locations $85 and $86. When storing, the routine checks for the end of RAM and displays the overflow flag if the end is reached. No more loading occurs but all further incoming data is displayed on the 25th line. Depressing the space bar terminates cassette input and reverts back to the keyboard for input. Anything typed now will be stored in the next location. At this point you should break to the monitor and call your EPROM loading routine. I use my smart terminal program to perform the storage function, but since the hooks are unique to my system, I mention only the following: transfer the memory image to your EPROM programmer and program the EPROM. I do

Listing 2

```
 10 0000                ;STORE, 1-1-82
 20 0000                ;BY DAVID A. JONES
 30 9600                *=$9600
 40 9600                ;
 50 9600                MEND  =$86    END OF MEMORY, HIGH ADDRESS BYTE
 60 9600                STOR  =$F0    STORE POINTER, 2 LOCATIONS
 70 9600                YTMP  =$FE    TEMPORARY STORE FOR Y REGISTER
 80 9600                LFLG  =$0203  BASIC LOAD FLAG
 90 9600                FLAG  =$D390  MODE FLAG
100 9600                BASIN =$FFEB  INPUT ROUTINE
110 9600                BASOUT=$FFEE  OUTPUT ROUTINE
120 9600                ;
130 9600 A003     SETP  LDY #$03    SET STORE POINTER
140 9602 84F1           STY STOR+1
150 9604 A000           LDY #0
160 9606 84F0           STY STOR
170 9608                ;
180 9608 20EBFF   CHECK JSR BASIN   GET CHAR FROM KEYBOARD
190 960B C94C           CMP #'L     CHECK FOR L TO START LOAD
200 960D D0F9           BNE CHECK   IF NOT KEEP LOOPING
210 960F 8D90D3         STA FLAG    DISPLAY FLAG
220 9612 A9FF           LDA #$FF
230 9614 8D0302         STA LFLG    SET BASIC LOAD FLAG
240 9617                ;
250 9617 20EBFF   IN    JSR BASIN   GET CHAR FROM ACIA
260 961A C900           CMP #0      DON'T STORE NULLS
270 961C F0F9           BEQ IN
280 961E 20EEFF         JSR BASOUT  DISPLAY CHARACTER ON CRT
290 9621 84FE           STY YTMP    SAVE Y
300 9623 A000           LDY #0
310 9625 91F0           STA (STOR),Y  STORE CHARACTER
320 9627 E6F0           INC STOR
330 9629 D008           BNE RETN
340 962B E6F1           INC STOR+1  INCREMENT BLOCK IN MEMORY
350 962D A5F1           LDA STOR+1
360 962F C586           CMP MEND    END OF MEMORY ?
370 9631 F005           BEQ OVER
380 9633 A4FE     RETN  LDY YTMP    RECALL Y
390 9635 4C1796         JMP IN      GET NEXT CHARACTER
400 9638                ;
410 9638 A94F     OVER  LDA #'O     DISPLAY OVERFLOW FLAG
420 963A 8D90D3         STA FLAG    ON 25TH LINE
430 963D 20EBFF         JSR BASIN   GET NEXT CHARACTER
440 9640 8D92D3         STA FLAG+2  DISPLAY ON 25TH LINE ONLY
450 9643 4C3896         JMP OVER    LOOP UNTIL BREAK
460 9646                ;
```

this with my machine-code file-save routine. This often is called the OSI checksum format but is really the MOS Technology Binary Loader, which is recognized by many commercial PROM programmers. The OSI Extended Monitor calls this the SAVE routine.

You may wonder why you should go to all this trouble when a disk offers more versatility. Some reasons are: cost, speed (a 3K BASIC program takes 3 minutes and 14 seconds — EPROM takes 13 seconds), ease of operation for novice users, absence of mechanical transports, the fact that the program is generally crashproof and quickly recovers if a crash occurs, and the need for less desk space and interface cables.

Some of the programs I find useful to have in EPROM (in addition to games) are a renumber utility, a word processor, a stock market monitor with a cassette data base that I update each week, and brief programs I might want on short notice. I installed a ZIF connector at location $4000 to facilitate changing PROMs.

You could, of course, have a completely dedicated system with PROMs as the only mass storage media. Such a system could be used in a process control environment or office that constantly uses the same set of programs day after day with no changes. The system could be programmed to boot and call a menu on power up. Changing the output vector to load the program without displaying it would speed up the loading process by about 50% — three or four seconds *versus* seven or eight seconds for a 4K program — and would be more asthetically pleasing to a non-computer professional.

For the do-it-yourselfer, three sources of EPROM programmer kits are listed here:

1. Micro Technical Products, Mesa, Arizona, $45.00 - kit, $15.00 - bare board, documentation and software.
2. John Bell Engineering, Redwood City, California, $39.95 - kit, $24.95 - bare board only.
3. Aardvark Technical Services, Walled Lake, Michigan, $75.00 - assembled and tested, $24.95 - bare board only.

Although the Aardvark board is the most expensive, it is designed specifically for the 600 board. The John Bell board is for the Apple but can be coaxed to work with any 6502 system. The Micro Tech unit is a general-purpose board and is the one I ordered.

Other Methods

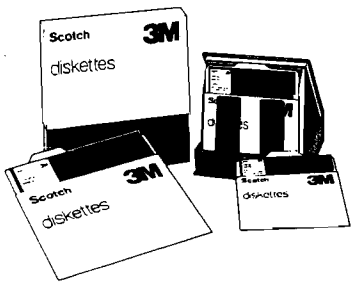
You can double the ROM/PROM capacity of the 600 board with or without the J1 connector. BASIC-in-ROM is stored in four 2316's which are 2K by 8-bit 24-pin ROMs in U9, U10, U11, and U12. Replacing these with 4K by 8-bit 2732 EPROMs doubles the storage capacity for a total of 16K available at these physical locations. The extra address decoding needed to support the additional 8K of EPROM requires only one 74LS139 IC and can be installed in one of the proto locations.

Assigning addresses $8000-$9FFF to the newly installed memory works well with existing usage and minimizes the possibility of conflict if you decide to add the 610 board later. BASIC-in-ROM starts at $A000; i.e., $9FFF + 1.

Now that you have this extra memory available, how can you use it to best advantage? Relocating the Extended Monitor to $9800-$9FFF keeps

Scotch® Diskettes

record reliability



at the lowest price!

Call our Modem Hotline (anytime) - 619-268-4488 for exclusive monthly specials. Our free catalog contains more than 600 fantastic values.

ABC Data Products
(formerly ABM)

8868 CLAIREMONT MESA BLVD.
SAN DIEGO, CALIFORNIA 92123

ORDERS ONLY 800-850-1555
ITT TELEX 4992217
INFORMATION 619-268-3537

CSE means OSI

Software and Hardware
Specializing in C1P and C4P machines

Basic Load/SAVE:
Employs token loader system. 50-100% faster than the old indirect ASCII system. Maintains a listing of file names found on the tape
C1P.....$10.95
C4P.....$19.95*

Basic Enhancer:
Renumber, Auto Sequencer, Screen Control functions, and tape I/0 system that is faster and has file names
C1P.....$21.95
C4P.....$29.95*

*comes with required modified monitor Rom chip

NEW! NEW! NEW!
ANCHOR SIGNALMAN MODEMS $95.00

Please write for more info on new disk programs or send $2.00 for catalog. Please include $2.00 shipping ($4.00 for modems).

Computer Science Engineering

Box 50 • 291 Huntington Ave. Boston 02115

it out of the user memory area and allows it to co-reside with the Assembler/Editor, increasing its usefulness. You can now jump back and forth between the Extended Monitor and the Assembler/Editor at will without reloading either program or source code. This makes debugging easy.

Since the original Extended Monitor resides in memory from $0800-$0FFF, the offset is exactly $9000, simplifying cross references to the original. A table located at $0960-$0999 is used to decode the monitor command and jumps to the selected routine. When you relocate the code, this table must be modified manually by adding the offset to every odd location within the table; i.e., $B30B becomes $B39B — the new jump address for the 'A' command at $0962,0963. The Monitor is now entered at $9800 and no user RAM is employed (page zero excepted).

I located the Assembler/Editor storage from $8040 to $9191. The Assembler/Editor uses self-modifying code and must be resident in RAM memory to run. I use the routine in listing 3 to move the code to its original location. Note that none of the text storage area ($1391 and up) is written to during the move, so recovery from

Listing 3

```
                        ;ASSEMBLER CALL, 11-15-81
                        ;BY DAVID A. JONES
                        ;RELOCATES CODE STORED IN ROM TO RAM
                        ;
                        *=$8000
                        FRHI=$80        FROM ADDRESS, HIGH BYTE
                        FRLO=$40        FROM ADDRESS, LOW BYTE
                        TOHI=$02        TO ADDRESS, HIGH BYTE
                        TOLO=$40        TO ADDRESS, LOW BYTE
                        FREG=$F6        TEMPORARY FROM ADDRESS STORE
                        TREG=$F8        TEMPORARY TO ADDRESS STORE
                        BLKS=$12        NUMBER OF BLOCKS TO MOVE (HEX)
                        ASBM=$1300      ASSEMBLER/EDITOR ENTRY POINT
                        ;
                        ;
8000 A980               LDA #FRHI
8002 85F7               STA FREG+1
8004 A940               LDA #FRLO
8006 85F6               STA FREG
8008 A902               LDA #TOHI
800A 85F9               STA TREG+1
800C A940               LDA #TOLO
800E 85F8               STA TREG
8010 A212               LDX #BLKS
8012 A000               LDY #0
8014                    ;
8014                    ;
8014 B1F6       BLOCK   LDA (FREG),Y
8016 91F8               STA (TREG),Y
8018 C8                 INY
8019 C052               CPY ##52        CHECK FOR POSSIBLE END OF CODE
801B F00B               BEQ TEST        DO REST OF CHECK IF SO
801D C000               CPY #0          IF NOT CHECK FOR END OF BLOCK
801F D0F3               BNE BLOCK
8021 E6F7               INC FREG+1      INCREMENT BLOCK POINTERS
8023 E6F9               INC TREG+1
8025 CA                 DEX             DECREMENT BLOCK COUNTER
8026 D0EC               BNE BLOCK
8028                    ;
8028 E001       TEST    CPX #1          IF X=1 THEN WE'RE FINISHED
802A D0E8               BNE BLOCK       OTHERWISE
802C                    ;
802C 4C0013             JMP ASBM
802F                    ;
```


RAM

For ATARI
48K RAM BOARD FOR THE 400
with Lifetime Warranty

- Highest quality available
- Reduces power consumption
- Reduces heat

48K Board (400) **$150**
32K Board (400/800) **$ 90**
16K Board (800) **$ 60**

FREE SHIPPING ANYWHERE IN U.S.A.

INTEC PERIPHERALS CORP
906 E. Highland Ave.
San Bernardino, CA 92404
VISA (714) 881-1533 MasterCard

ATARI, 400, 800 are Trademarks of ATARI, Inc.

OSI Disk Users

Double your disk storage capacity Without adding disk drives

Now you can more than double your usable floppy disk storage capacity—for a fraction of the cost of additional disk drives. Modular Systems' DiskDoubler™ is a double-density adapter that doubles the storage capacity of each disk track. The DiskDoubler plugs directly into an OSI disk interface board. No changes to hardware or software are required.

The DiskDoubler increases total disk space under OS-65U to 550K; under OS-65D to 473K for 8-inch floppies, to 163K for mini-floppies. With the DiskDoubler, each drive does the work of two. You can have more and larger programs, related files, and disk utilities on the same disk—for easier operation without constant disk changes.

Your OSI system is an investment in computing power. Get the full value from the disk hardware and software that you already own. Just write to us, and we'll send you the full story on the DiskDoubler, along with the rest of our growing family of products for OSI disk systems.

™DiskDoubler is a trademark of Modular Systems.

Modular Systems

Post Office Box 16C
Oradell, NJ 07649.0016
Telephone 201 262.0093

moderate crashes can be made by calling the move routine again and resetting the 'next available storage location pointer' to the end of the text file +1. This pointer is stored in location $12FE,12FF and is set to $1391 upon initialization. Cold entry to the Assembler/Editor is from $8000 and the 'INIZ' prompt is displayed instantaneously. Warm entry is from $1300 and you should answer 'no' to 'INIZ', as before.

Putting only these two utilities in PROM is worth the cost and effort, but there is more. PROM space from $9191 to $9800 is available for custom routines. I have my printer interface driver here, a new keyboard service routine, Autobasic, a smart terminal routine, an EPROM programmer, and still have space left over for future expansion.

There's another advantage! By making BASIC-in-ROM BASIC-in-PROM, you can fix the garbage collect bug that exists in Microsoft BASIC Version 1, Rev. 3.2. I have implemented the code recommended by E. Morris in the June '81 issue of *PEEK(65)* and am pleased with the results. In fact, the word processor I used to write this article is written in BASIC and uses extensive string manipulation.

The diagrams and schematics accompanying this article show you how to make the necessary modifications. Most of the cuts to the PC traces are on the top of the board and are marked with a —ʃʃ—. Figures 3 and 4 show the best places to make the cuts.

The jumpers (not shown in figure 3) should be apparent in figure 2. I marked the source of each added or changed signal and abbreviated the schematic for clarity. You should compare it with the original before you attempt the

***Figure 1:* Chip Select Decoding**

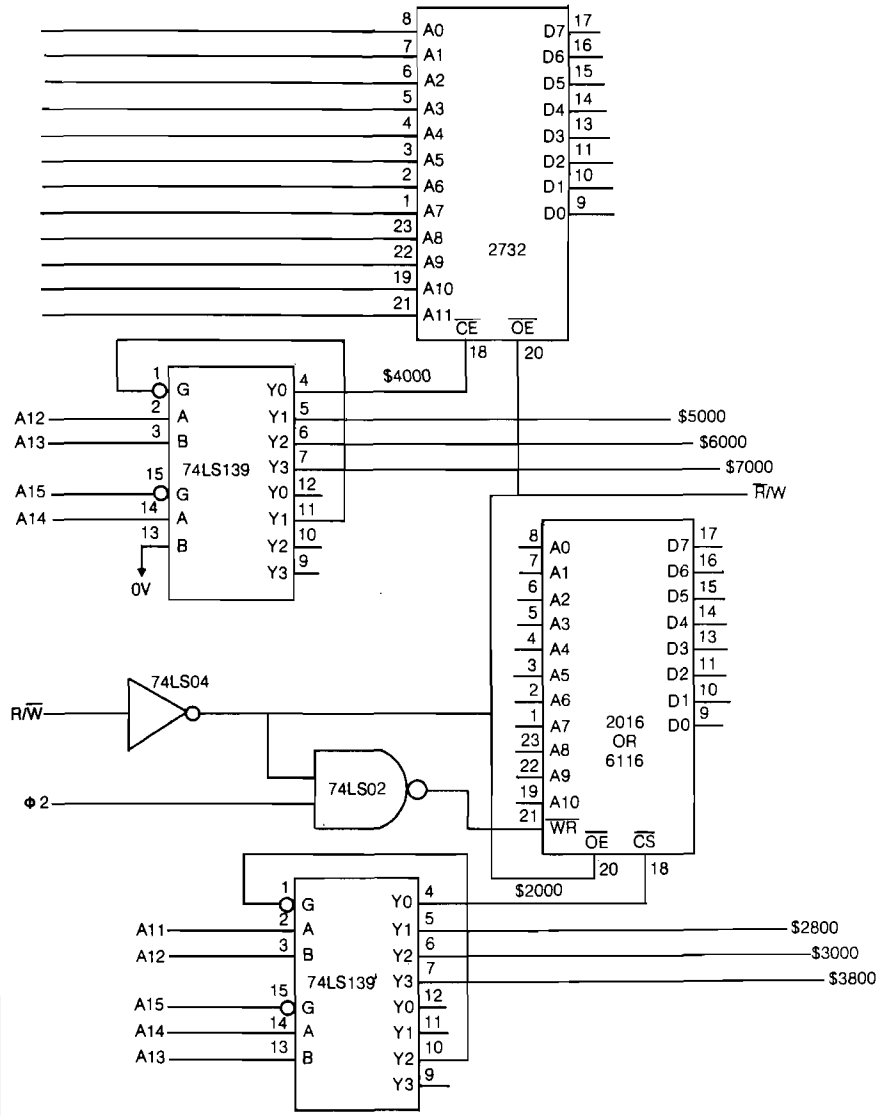


Figure 2

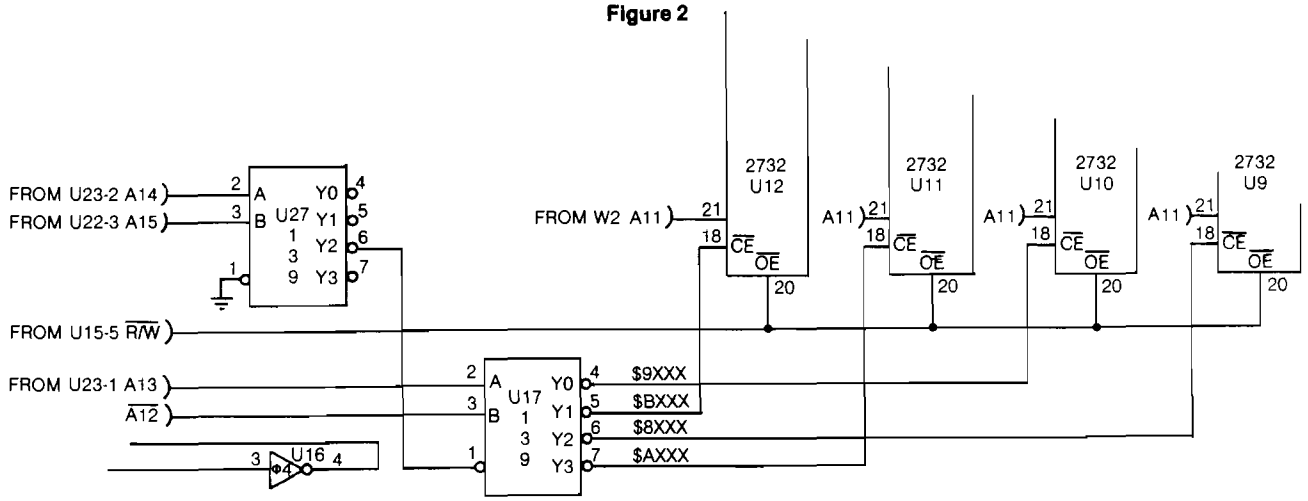


Figure 3: **Top View of PC Board**

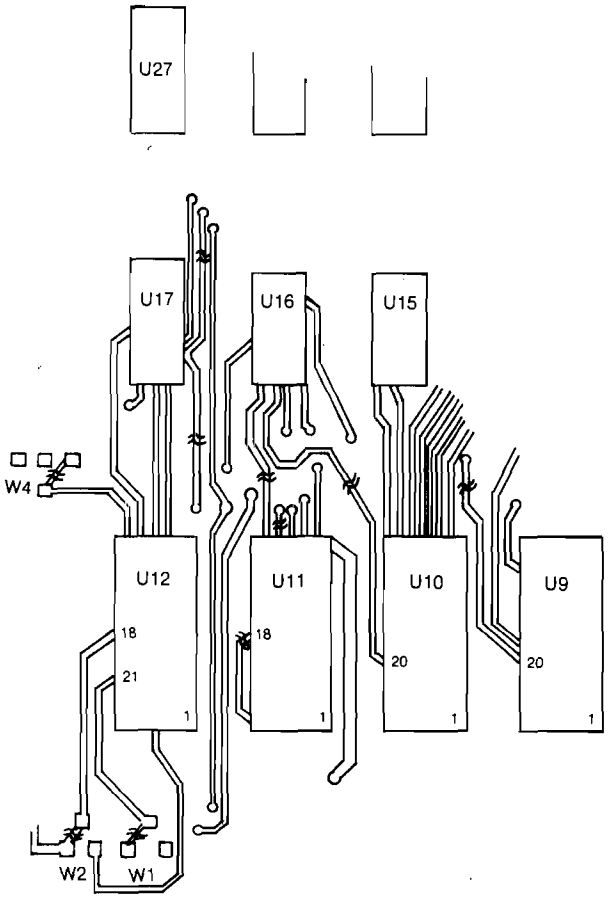
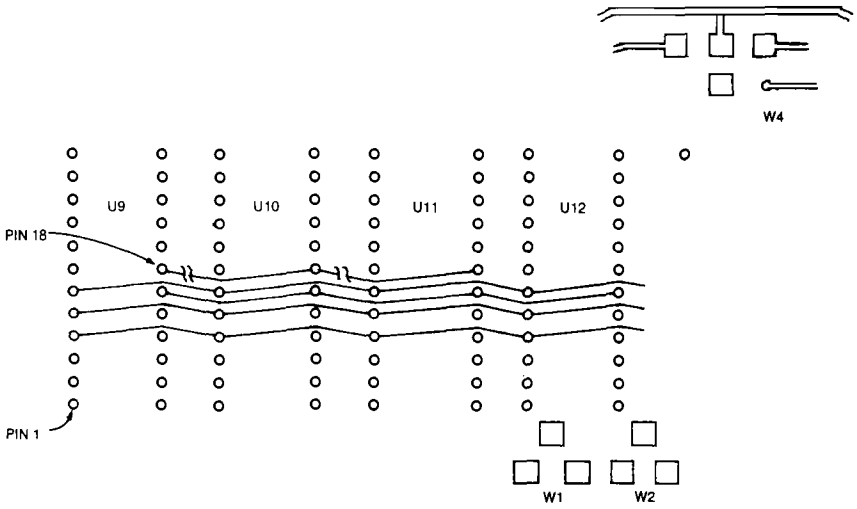


Figure 4: **Bottom View of PC Board**



modification. Basically, all that is necessary for the conversion is to remove 02 from pin 21 and replace it with A11, wire in new output enables to pin 20, and add the new chip selects on pin 18.

The 2316's use high logic levels for CS and therefore require the four sections of inverter U16. The 2732's use low logic levels for CS or CE, so the inverters are not used.

You should have confidence in your ability to make hardware modifications before attempting these changes. The entire project can be done in one evening and will provide you with increased performance from your Superboard II/C1P at moderate cost.

David Jones is a technical specialist for Racal Milgo Inc., a Miami-based manufacturer of data communications products. He has a Superboard II and has redesigned the video circuitry for a true 32/64-character per line display, built a parallel printer interface, adapted an EPROM programmer *via* a 6522 VIA, and customized the monitor program in addition to the expansion outlined in this article. You may contact Mr. Jones at 8902 S.W. 17 Terrace, Miami, FL 33165.

MICRO

OHIO SCIENTIFIC

NEW PROGRAMS!

SCOUT—Full color, machine language, fast action and graphics! After a year of development, comes the all machine language **SCOUT.** Patrol the planet surface protecting and saving the human population from abductors. Turn your OSI into a real arcade!
$24.95 C4PMF, C8PDF.

Send for our FREE catalog. We have what you want for less: S-FORTH $39, FULL SCREEN EDITOR $19, ADVENTURE $19, SKYHAWK $8, TOUCH TYPING $19, INTELLIGENT TERMINAL $24, THE WIZARD'S CITY $12, UTILITIES, and much more for the C1P to the C8PDF.

(312) 259-3150
AURORA SOFTWARE
37 S. Mitchell
Arlington Heights,
Illinois 60005

Indirect Files Under OS-65D

by Richard L. Tretheway

Several uses for the indirect files are discussed, including merging programs, a warm start, and transferring programs from one OSI operating system to another.

Demo Programs
requires:
OSI with OS-65D V.3.3

The indirect file is one of the most powerful and yet underused features of Ohio Scientific's OS-65D operating system. This file is a buffer in memory that can be used for temporary storage of either BASIC or assembly-language programs.

You would use an indirect file, for example, when you are in the midst of writing a program and suddenly realize that you either haven't created a disk file for the program at all, or the created file is no longer large enough to hold the current version. At that point, you could either store the program in your scratch file (that nobody seems to remember to have ready for such purposes), or you could use the indirect file. To send your program to the indirect file just enter the command "LIST <SHIFT> 'K' ". You will see the program listed normally, but it is also being stored in the indirect file buffer. When BASIC returns the "OK" prompt you enter a <SHIFT> 'M', which marks the end of the file in memory. Your program workspace is now free to run the directory program, or "CREATE" or "DELETE" as needed. When you have created the proper file, enter "NEW" to clear the workspace and type a <CTRL> 'X'. Your program will be reloaded into the workspace intact. Don't be concerned when you enter the <SHIFT> 'M' and BASIC returns a syntax error. The operating system knows what you want done.

Those of you familiar with indirect files probably noticed that I made no mention of the POKEs to 9554 and 9368, which set the starting addresses for input and output for the indirect file. OS-65D was developed on and for systems with 48K of memory and thus these addresses are set by default to $8000 hex. If you don't happen to have 48K, you will have to reset these addresses with the appropriate POKEs. One good way to decide what number to use is to figure out how many pages of workspace memory you have, divide it by two, and then subtact that number from the page number of the end of your contiguous memory. For example, on a 24K system running OS-65D V3.2 you have roughly 12K of free RAM, which translates to 48 pages of memory. Dividing that by 2 yields 24. The top of 24K is page 96. 96 – 24 = 72 and thus, 72 is what I recommend you POKE 9554 and 9368 with in your BEXEC* so that it is always set up and available when needed.

If you own an OSI disk system manufactured before August of 1980, you have probably cursed OSI's fast

Listing 1

```
10 REM-  Program to add DATA statements to existing programs
20 REM-          to set up USR(X) functions.
30 S=INT((PEEK(8960)-59)/2)+59:POKE133,S-1
40 SS=PEEK(8960):REM- System Memory Size in Pages.
50 PRINT!(28)"To begin, I need to know the starting and ending"
60 PRINT"memory address where your machine code is in memory.":PRINT
70 PRINT"For the starting address,":GOSUB490:SA=A
80 PRINT"For the ending address,":GOSUB490:EA=A
90 PRINT:INPUT"Was this code assembled with an offset";Y$
100 IFLEFT$(Y$,1)<>"Y"THENPRINT:OF=0:GOTO120
110 PRINT:PRINT"For the offset,":GOSUB490:OF=A
120 NL=INT((EA-SA)/15)+1:REM- # lines = size/15 + 1
130 SP=NL*52:REM- Space = Approx. 52 bytes * Number of lines.
140 IF EA+SP < SS*256 THEN A=(INT(EA/256)+1)*256:GOTO250
150 IF S*256+SP < SA THEN A=S*256 : GOTO250
160 IF (EA-SA)+SP < (SS*256) THEN200:REM- Will it ever fit ?
170 PRINT!(28)"This code needs to be reassembled higher in"
180 PRINT"memory in order for it to fit in your available"
190 PRINT"RAM along with the generated indirect file.":END
200 PRINT"This machine code routine is simply too large to fit in"
210 PRINT"memory along with the generated indirect file."
220 PRINT"You might try breaking it up into pieces and then"
230 PRINT"re-run this program on each individual piece merging"
240 PRINT"the results into a large program on disk."
250 POKE9554,A/256:POKE9368,A/256:REM- Set up indirect file.
260 DA=A:SA=SA-OF:EA=EA-OF
270 PRINT"What line number should the DATA reading"
280 INPUT"start with ";LN:PRINT
290 INPUT"How much should I increment each line number ";I
300 POKEDA,13:POKEDA+1,10:DA=DA+2:REM- Initial <CR><LF>
310 O$=STR$(LN) : GOSUB600: POKEDA,32 : DA=DA+1
320 O$="FORX=" : GOSUB600: O$=STR$(SA) : GOSUB620: O$="TO"
330 GOSUB600: O$=STR$(EA) : GOSUB620:POKEDA,58 : DA=DA+1
340 O$= "READY:POKEX,Y:NEXTX:RETURN" : GOSUB600
350 POKEDA,13 : POKEDA+1,10 : DA=DA+2 : A=SA+OF
360 LN=LN+I : O$=STR$(LN) : GOSUB600: DC=0
370 O$=" DATA" : GOSUB600
380 O$=STR$(PEEK(A)) : GOSUB620
390 PRINT".";: IF POS(0)=63 THEN PRINT!(12)
```

(Continued)

<BREAK> key more than once. One of the foibles of OS-65D is that there is just no good way for you to do a warm start after the system has been reset by a <BREAK>. But fortunately, OSI didn't really desert you after all. The indirect file can save all of your hard work. Try booting your system and enter BASIC with your "BEXEC*" program still loaded in the workspace. Now press the <BREAK> key. At the "H/D/M" prompt, type "M" to enter the machine code monitor, and then press "G" for GO. You should now be back in BASIC, albeit a crippled BASIC. As previously described, enter "LIST <SHIFT> 'K' ", followed by a <SHIFT> 'M' at the "OK" prompt. Now re-boot the system. Clear the workspace with the "NEW" command. This time at the "OK" prompt type a <CTRL> 'X'. Your program has been re-loaded, intact, into the workspace all ready to be run and/or saved to disk. While I'm sure that this is a patch that was added after BASIC had been written for OSI systems, it works and has saved me more hours than I'd care to admit.

You can also use indirect files to transfer programs from one operating system to another. One of the ways that OSI protects users from accidentally trying to run incompatible software from one operating system to another is to use different disk formats so that one DOS won't read files from another. If you own a color video system try booting up your "DEALER DEMO" disk, press "P" for "PASS" to enter BASIC. Now insert a regular OS-65D disk in the drive and try to load the "BEXEC*" from that disk. No soap, right? Here again, indirect files can help. All versions of OS-65D support indirect files as does OS-65U. So, to transfer programs from one OS to another, load your program and send it to the indirect file. Re-boot on the destination disk. Type "NEW" to clear the workspace and type a <CTRL> 'X', and you're all set to go.

Indirect files can merge programs. Assembly-language programmers are notorious for having little pet subroutines that they use in many different programs. If you are careful to use line numbers that are dedicated to special functions, you can use indirect files to avoid having to retype those subroutines every time you need them.

To illustrate a use for indirect files while in the Assembler/Editor, try loading an assembly-language program. Now type "P <SHIFT> 'K' ". As with BASIC, you will see your program listed on the screen. Type a <SHIFT> 'M' to close the file. Now type "I" and respond with a "Y" to the prompt "INIZ?". Confirm that the workspace is clear by typing "P" again. Now try our usual <CTRL> 'X'. Your program will be reloaded. The operating system stores your program in discreet ASCII when you use the indirect file. When you download a program from the indirect file, the OS interprets the incoming lines just as if you had typed them in from the keyboard. Should you have a program in the workspace while you are downloading a program, duplicate line numbers will be replaced by the new lines just as if you had typed in a correction to a line with a syntax error in it.

But a crafty assembly programmer could arrange a library of disk files containing his most used subroutines, being careful to avoid duplicate line numbers. Whenever one of those progams is needed, he would just send the destination program that was resident in the workspace to the indirect file, then load the subroutine file and merge the two by typing a <CTRL> 'X'. It's not perfect, but at least it can save a lot of typing.

To illustrate another use of indirect files, I have written a short program. This program takes a machine-code program (resident in memory) and converts it into a subroutine written in BASIC that does the appropriate POKEs to set up a USR(X) function. It could be used to transfer a machine-code program from your disk system to a friend's system that just has a cassette. He still won't be able to read your disk, but at least he would have a copy of your program in a form he can use.

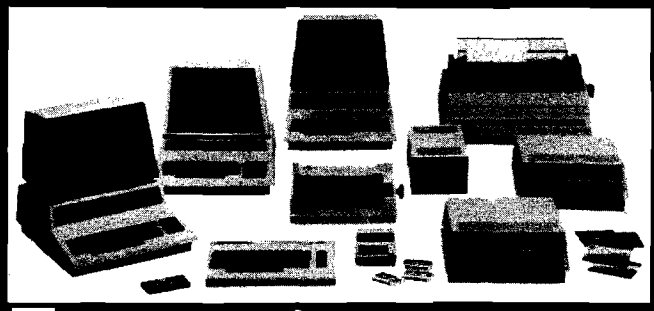
To use the program, save it on disk and then assemble your machine-code program in memory. Since the memory maps of disk and cassette systems are very different, my program takes any offsets that you had to use into account when creating the new subroutine. When the machine code is in place, re-enter BASIC and run my program. It will ask you where the machine code is in memory (and will accept the hex address if you didn't figure it ahead of time), what line numbers to use for the subroutine to be created, and finally, offsets, if any. After the program has run just type a <CTRL> 'X', since my program has already NEWed itself out of existence. You can now save this program on disk, or run it and it will faithfully recreate your machine-code program. Note that my program was written using OS-65D V3.3 and has some print statements that do cursor addressing under that OS. Those of you still running 65D V3.2 can eliminate the parts of the print statements that do this (i.e., "![12]" in line 280 should be eliminated).

Listing 1 *(Continued)*

```
400 IF A=EA+OF THEN 450: REM- Done ?
410 IF A>SS*256 THEN 160 : REM- Out of RAM ?
420 A=A+1 : DC=DC+1 : REM- INC Address & Data Counter
430 IF DC<15 THEN POKEDA,ASC(",") : DA=DA+1 : GOTO380
440 POKEDA,13 : POKEDA+1,10 : DA=DA+2 : GOTO360
450 POKEDA,13:POKEDA+1,10:POKEDA+2,93:POKEDA+3,13:POKEDA+4,10
460 PRINT!(28)"The subroutine for your USR(X) routine now resides"
470 PRINT"in the indirect file. To add it to your program, load"
480 PRINT"your program into the workspace and type a <CTRL> 'X'.":NEW
490 PRINT"Enter the decimal address or hex address preceeded by a '$'"
500 INPUT" ";A$ : PRINT : L=LEN(A$) : IF LEFT$(A$,1)="$"THEN530
510 FORX=1TOLEN(A$):C$=MID$(A$,X,1):IFC$<"0"ORC$>"9"THEN490
520 NEXTX:A=INT(VAL(A$)):RETURN
530 A=0:IFLEN(A$)<2THEN490
540 FORX=2TOLEN(A$):C$=MID$(A$,X,1):IFC$<"0"THEN490
550 IFC$<="9"THENA=A+VAL(C$)*(16^(L-X)):GOTO580
560 IFC$<"A"ORC$>"F"THEN490
570 A=A+(ASC(C$)-55)*(16^(L-X))
580 NEXTX
590 RETURN
600 FORX=1TOLEN(O$):POKEDA,ASC(MID$(O$,X,1))
610 DA=DA+1:NEXT:RETURN
620 O$=RIGHT$(O$,LEN(O$)-1):GOTO600
```

You may contact the author at 5405 Cumberland Rd., Minneapolis, MN 55410.

SJB DISTRIBUTORS. THE MOST COMPETITIVE PRICES ON COMMODORE PRODUCTS.



commodore

INTERFACES

ADA-1450 Serial	$149
ADA-1600 Parallel	149
RS232 cable for Vic or 64, 2m	30
Video/Audio cable for 64 & monitor	25

MONITORS — Great resolution for the CBM 64 or VIC

Panasonic, 13" Color	$375
Amdek Color I	330
NEC JB 1201M, 12" Color	330
NEC JB 1201, 12" green phosphor	170
Amdek Video 300L, green phosphor	175

BUSINESS SOFTWARE

Spellmaster Dictionary (great for WordPro!)	$199
OZZ Data Base System (8050)	240
Silicon Office (database, wp)	995 (New)
Wordcraft 80	289
VisiCalc (new expanded)	199
Dow Jones Portfolio Management System (RS232)	120
WordPro 4 + or 5 +	299
The Manager	199
Legal Time Accounting	425
I.R.M.A.	295
BPI A/R, G/L, Job Cost, Inventory, Payroll	325 pkg

SJB will service any VIC or CBM64.

MasterCard, Visa, Money Order, Bank Check
COD (add $5) accepted.
Add 3% surcharge for credit cards.
In stock items shipped within 48 hours, F.O.B., Dallas, TX.
All products shipped with manufacturer's warranty.

TO ORDER CALL TOLL FREE
800-527-4893 800-442-1048 (Within Texas)
SJB will meet any competitive price under similar in-stock conditions.

SJB DISTRIBUTORS, INC.
10520 Plano Road, Suite 206
Dallas, Texas 75238
(214) 343-1328

Business Hours:
M-F 8 to 6
Sat 10 to 2

Prices are subject to change without notice.

SOFTWARE FOR CBM 64

Word Processing	$90
Computer Tutoring Game (COCO)	50
General Ledger	199
Pet Emulator (emulates 4.0 basic)	30
CBM EasyCalc (for the 64)	99
CBM EasyFinance	50
CBM EasyPlot	80
CBM EasyScan (appointment manager)	80
Sprite-Magic (build sprites on screen with Joystick, save to disk or cassette)	30
Assembler Package for CBM 64 (cassette) Editor (creates and updates source code), Assembler, Loader, Disassembler	50
Mail Mate	50
IEEE Interface (64)	100
Parallel Interface	90
RS232 Interface (modems, printers)	45

VIC PRODUCTS

VIC 20 Computer, 5K	$199
Vic Datasette Recorder	60
Vic 1541 Disk Drive	395
VIC MODEM (for CBM 64)	100
VIC 1525 Graphic Printer (for CBM 64)	325
8K Memory Expansion Cartridge	49
16K RAM	99
24K RAM	155
IEEE Interface (VIC)	85
Gorf (great arcade game)	30
Omega Race	30
Midnight Drive	23
VIC 3 slot Expander	43
VIC 6 slot Expander	83
Seawolf	23
Cosmic Cruncher	23

Arcade Joysticks — Heavy duty with 2 firing *buttons! Great for the VIC or 64* **$25**

SuperPET (5 languages, 2 processors)	$1409
CBM 8032 Computer, 80 column	1029
CBM Memory Expansion, 64K	359
PET 4032, 40 Column	950
CBM 8050, 1 Mg. Dual Drive	1259
CBM D9060, 5 Mg. Hard Disk	2240
CBM D9090, 7 5 Mg. Hard Disk	2600
CBM 4040, 340K Dual Drive	919
CBM 2031, 170K Single Drive	489

PRINTERS — LETTER QUALITY

CBM 8300,40cps	$1450
Diablo 620,25cps	1350
Nec Spinwriter 7700,55cps	2350
Nec Spinwriter 3500,35cps	1600

PRINTERS — DOT MATRIX

CBM 4022,80cps graphics	$395
CBM 8023,150 cps graphics	599
Okidata 82A,120cps serial or par	449
Nec 8023A(parallel)	499

MICRO™

From Here To Atari

By Paul S. Swanson

Languages

The language C, offered in interpreter and compiler versions, was recently added to the list of languages available to Atari 400 and 800 systems. FORTH, PILOT, and several other high-level languages have been available for some time. But despite the large number of choices for languages, the most popular seem to be BASIC and assemblers.

There are several versions of BASIC and several assemblers. Of the assemblers, the original cartridge version from Atari can be implemented on even a 16K system, which is an advantage for those who have not expanded their systems to 48K. Atari also provides a macro assembler. I chose the Synassembler (Synapse Software), which assembles faster than the Atari cartridge and is less expensive.

Almost every Atari computer is purchased with a version of BASIC — usually the cartridge BASIC. There is also a form of Microsoft BASIC for those of you who want to, on your Atari, run software that was written for other computers. Because of the note and point style of random access used on the Atari, there is a big difference in the disk commands between the Atari version and the versions on the Apple, Radio Shack, and IBM computers.

Monarch Data Systems (P.O. Box 207, Cochituate, MA 01778) offers a BASIC compiler. Compiled programs can run 4 to 12 times faster than the interpretive code. Several restrictions limit the use of this compiler, but these are easily circumvented for most applications. For example, the compiler uses fixed point arithmetic instead of the slower floating point, so there are no trigonometric functions. There is also no RND(function supported, so you must use a PEEK(53770) and extract a random number by multiplying that result (a random number between 0 and 255), then dividing to get a number in the proper range. The LOAD and RUN statements are also not supported, so the program cannot chain to other programs. For more information on this compiler, contact Jeff Goldberg at Monarch Data Systems.

Hardware

Many letters from readers express an interest in the hardware and ask questions about the keyboard I was marketing. I recalled the keyboard because several manufacturers produce keyboards just like it, and they can produce them more cheaply than I can. However, I am writing an article describing how I built my keyboard (you can build one for under $20 plus a weekend of time instead of buying one for over $100).

Several questions have been asked about the controller jacks at the front of the Atari console. These jacks provide the simplest interfaces to any external device. They are connected to eight A/D (analog to digital) converters and a PIA (peripheral interface adapter). See the last page of your Atari hardware manual for a diagram of the pins and jacks. Port A controls the joystick pins of jacks 1 and 2 and Port B controls those on jacks 3 and 4. As you look at the computer from the front, the MSB is on your right and the LSB is on your left. To use them for input, you do not need special codes — just PEEK(54016) for Port A or PEEK(54017) for Port B.

To set up pins for output, you must write to the direction control register. An example of setting up Port B as an 8-pin output follows: (to use Port A instead, subtract one from the PEEK and POKE addresses)

```
100 POKE 54019,56
110 POKE 54017,255
120 POKE 54019,60
```

POKEing 54019 with a 56 tells the computer to take the next POKE to 54017 as a direction control code. This is a binary code with each bit corresponding to a pin on the jacks. If the corresponding bit is 1, the pin is defined as output and if it is a zero, the pin is set up as input.

Once you have completed that section of code, you may then POKE to 54017 whatever you want to send out. If you POKE there and then PEEK the same location, you will get back the code you sent, as if it were a RAM location. Therefore, if you set the low-order four bits as output and the upper four as input, you can send a code out then read the input combined with the code you sent. This makes scanning controllers simple to set up in the software. The value you read is what you sent plus 16 times the value that your device sends back.

The plugs for those jacks are not easy to obtain. You can get plugs that work from APX, but they cost almost $7 each (plus postage) and you must have a minimum order. Just check around in your area for a suitable store. My source is Eli Computers in Cambridge, MA.

A ground and +5 volts are available, also. For larger projects it is best to have an independent power supply for your device. According to Atari, you can draw as much as 300 mA from these pins (total — not per port), which should be enough to drive many smaller devices.

Thanks to Devin MacAndrew for calling my attention to an error in my November column. The 64K board I mentioned, according to the advertisement he sent me, is available from Mosaic (Mosaic Electronic Inc., P.O. Box 748, Oregon City, OR, 97045) and bank selects only above the 48K boundary, using a 4K address space not used by the hardware registers or operating system.

Future Columns

As many columns as possible will be based on mail I receive, so by all means, keep writing. Please mention specific applications and include a description of your hardware configuration. The next few columns will deal with various aspects of the hardware available on standard Atari computers.

MICRO™

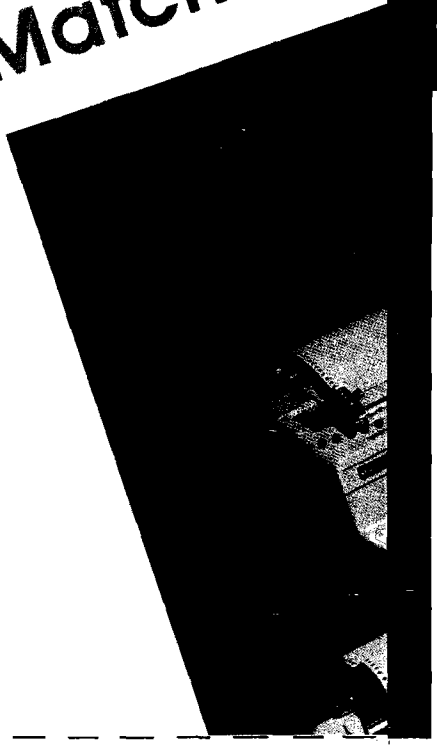
In The Beginning Was The Word...

MICROCOMPUTING®

October 198
USA $2.95 (UK£2
Number 7

A WAYNE GREEN PUBLICATION

Discover Sp
Matchless



MICROCOCCUS•MICROLITER

micrococcus, mi kro kok' us, n. a microscopic organisr of a round form.
Microcomputing, mi' kro kom put ing, n. (Gr. mikro small, and L. computo, to calculate.) The multi-syster monthly journal for computer enthusiasts, containin all the information needed to turn your microcompute into a powerful machine. Includes dozens of new pr grams, articles on innovative computer application buyer's guides, new programming techniques, accurat reviews of hardware and software, complete coverage new products, tips on your system's hidden capabilitie hardware modifications, tutorials, utilities, boo reviews, industry news. Plus features on computers i business, science, education and games. Written i understandable language by experts in the field of com puting. Special emphasis is placed on the Apple*, Atar Commodore, Heath and IBM systems, but not to the e clusion of other systems.
(Ed. note—A one year subscription to MICROCOM PUTING is only $24.97. Call

1-800-258-5473

Or send in the coupon below.)
microcopy, mi' kro kop i, n. A photographic copy printed material or photographs...

MICROCOMPUTING

The First Word in Computer Publishi

*Apple® is a registered trademark of Apple Computer, Inc.

YES! I want to get the First Word in Computer Publishing. Send me 12 issues of MICROCOMPUTING for $24.97. ☐ Check enclosed ☐ MC ☐ VISA ☐ AE ☐ Bill me

731 RMC

Card#_____ Exp. Date _____
Interbank# _____
Name_____
Address _____
City _____ State_____ Zip_____

MICROCOMPUTING®
Box 997
Farmingdale, NY 11737

Canada & Mexico $27.97, 1 Year Only, U.S. Funds
Foreign Surface $44.97, 1 Year Only, U.S. Funds Drawn on
Allow 6-8 weeks for delivery.

It's All Relative — Using CBM's Relative Records, Part 3

by Jim Strasma

This third installment describes the use of a key file as an index into a relative file. The author draws examples from the update module of a powerful mail-list package (available from the author and various user groups).

Last month we learned how to create relative files, using the "Create" module of Chris Bennett's "Mail List 4040." This time we will set up a key file as a framework for accessing relative files. Our example is the "Update" module of Chris Bennett's mail list. As before, you may want it handy as you read.

Cautions

Bennett's "Update" module, loaded from "4040 menu," is the largest module in the package. Be careful about changing it. If you lengthen it more than a couple of lines, you will need to increase the value POKEd into location 43 (or your BASIC's equivalent) in line 1060 of the "Setup" module, as explained in part one of this series (55:37). If you want to save space for more names, shorten this module and change the above line to match.

As supplied, there is only enough main memory free to maintain about 1200 names, even in disk drives far larger than the 4040. This is also about the size limit for a relative file on the 8050 disk drive; (i.e., six-side sectors referencing 120 sectors, containing 254 bytes of data each). The 8050 file size limitation does not apply to other Commodore disk drives.

Using a Key File

A relative file permits simple access to any record within a file, as long as we know its position within the file relative to other records. However, that is rarely the case. We are more likely to know the name we want than the record number. To deal with this, most file-handling programs use one or more additional files as indexes to the main file. These files usually contain a value for each record in the main file. The values are ordered in some way, usually reflecting the alphabetic ordering of some field within the records of the main file. Fields used this way are called key fields, because they act as keys to gain access to the file. In Bennett's mail list, the key field is a combination of two fields — the first ten characters of the last name, followed by two characters of the first name.

In smaller programs, the key file may actually contain the contents of the key field, along with the number of its record in the main file. This method provides the fastest access, but uses memory quickly. Therefore, large programs usually store the key value on the disk, accessing it there when needed. Either way, the key file is kept sorted at all times by the contents of the key field. This allows records to be located with a binary search (see Alfred Bruey's article on page 37 for more on binary searches), a technique offering very quick access to information already sorted. Here is the binary search used in the mail list:

```
4460 FS=0
...
4580 I=1
4590 J=NV
4600 IF I>  J THEN 4700
4610 K=INT((I+J+1)/2)
4620 I%=K%(K)
4630 GOSUB 5320:REM READ KEY
4640 IF K$=KY$ THEN 4680
4650 IF K$<KY$ THEN J=K-1:
     GOTO 4600
4660 K=K+1
4670 GOTO 4600
4680 RR=K%(K)
4690 FS=1
4700 RETURN
```

Variable FS starts equal to zero and is changed to +1 if key value K$ exists in the key field. Notice that array K%( ) contains only the numbers of records in the main file. Subroutine 5320 returns the key field from the main file in variable KY$:

```
5310 REM READ THE KEY IN RELATIVE
     RECORD NUMBER I%
5320 RECORD#1,(I%)
5330 IF DS THEN 1690
5340 INPUT#1,KY$
5350 IF DS THEN 1690
5360 RETURN
```

If you don't have BASIC 4, you will need to substitute for the RECORD# statement above (as described in part two of this series (56:52).

By making the key field the first one in each record, it can be read with a single INPUT# statement. Note that the only difference in BASIC 4 between reading a field from a relative file and reading it from a sequential one is the RECORD# statement preceding the read.

Keeping Keys Sorted

Next, let's consider the way keys are kept sorted. At any moment there is no more than one key value out of order — the one currently being added. When a record is to be added to the file, its key field is first checked, using the binary search above, to see if it is a duplicate of one already in the file. This is not allowed in the mail list, though some programs do permit duplicate keys. Next, another subroutine makes room for the new key in the K%( ) array that will become the updated index file at the end of the program run:

```
2360 REM DO BINARY INSERT
     ON KEY K$
2370 IF P<1 THEN P=1
2380 IF NV=1 THEN K%(1)=1:GOTO
     2490:REM EXIT
```

```
2390 EZ=P+2:IF EZ > NV-1 THEN
     EZ=NV-1
2400 FOR K=P TO EZ
2410 : I%=K%(K)
2420 : GOSUB 5320:REM READ KEY
2430 : IF K$ < KY$ THEN P=K:K=EZ
2440 NEXT
2450 IF K$ > =KY$ THEN K%(NV)=
     RR:GOTO 2490:REM EXIT
2460 E=NV
2470 SYS DL,0,P,E,K%(0),ZZ
2480 K%(P)=RR
2490 RETURN
```

Lines 2370-2390 and 2450 handle the top and bottom limiting values. The other lines perform a binary insert to find quickly the place where the new key field should go in the file. Once located, the SYS call in line 2470 opens a space for it in the K%( ) array. Until we cover these calls at the end of this series, readers without BASIC 4 may use a BASIC substitute:

```
2470 FOR QQ=E TO P+1 STEP -1
2473 : K%(QQ)=K%(QQ-1)
2476 NEXT
```

Where QQ is simply an otherwise unused variable. Once a hole is opened, it is filled with the value of the current record number RR.

When a record is deleted, the process is similar. First, its key and record number are found using the binary search above. Then, after double-checking that the user really wants to delete that record, its place in the key array is removed, with each value above it moving down one:

```
3250 IF P<> NV THEN SYS
     DL,1,P,E,K%(0),ZZ
3260 K%(NV)=0
```

Or, in BASIC:

```
3250 IF P=NV THEN 3260
3252 FOR QQ=P TO E-1
3254 : K%(QQ)=K%(QQ+1)
3256 NEXT
3260 K%(NV)=0
```

This process frees a space in the relative file for re-use. To handle this, Bennett's mail list remembers the deleted record number, RR, in a separate array DE%( ), indexed by ND, the number of currently deleted records.

```
3270 ND=ND+1
3280 DE%(ND)=RR
```

When new records are added later, these spots are re-used first, as we will see next time. As supplied, Bennett's mail list dimensions DE%( ) to NR, the maximum number of records. If you were suddenly to delete every one of the list's 1000 name capacity, the array would hold it. On the other hand, if you want to have over a thousand records, you may need to reduce the capacity of this array. Unless your list is highly volatile, a dimensioned size of NR/10 should be adequate. Array sizes are defined in line 2070 of the set-up module.

Reading the Key File

To be useful, key file information has to be in memory during the program run and on diskette, preserved for future use. Further, when its information is changed, the disk copy needs to be replaced. Naturally, we want to do all this as quickly, compactly, and safely as possible.

The first aid to speed is that the key file is read only once per session; if the currently needed key is already in memory from earlier use of this or another module, it is not read again. Variable GD (got data) keeps track of this for us:

```
1160 IF GD< >1 THEN GOSUB
     4720:GD=1: REM READ IN KEY
     FILE IF HAVEN'T
```

Since we will always want to read the entire key file at once, and in order, it is kept in a sequential disk file. When it is to be read, this routine does the work:

```
4710 REM READ IN KEY FILE
4720 DOPEN#9,D(DD),"INDEX" ON
     U(UN)
4730 IF DS THEN 1690
4740 INPUT#9,F$,NR,NV,ND
4750 IF DS THEN 1690
4760 IF NV=0 THEN 4790
4770 FOR I=1 TO NV:INPUT
     #9,K%(I):IF DS=0 THEN NEXT:
     GOTO 4790
4780 GOTO 1690
4790 IF ND=0 THEN 4820
4800 FOR I=1 TO ND:INPUT#9,
     DE%(I):IF DS=0 THEN
     NEXT:GOTO 4820
4810 GOTO 1690
4820 DCLOSE 9
```

In BASIC 2, substitute:

```
4720 OPEN#9,UN,9,DD$+"INDEX,
     SEQUENTIAL,READ"
4725 GOSUB 60010:REM CHECK DISK
     STATUS
     (also use as line 4745, and within
     lines 4770 and 4800, just before
     IF DS.)
4820 CLOSE 9
```

Line 4720 opens the sequential file "index" to read. Then a few special values are read in:

F$, the name of the mail list
NR, the next record number to be used
NV, the maximum number of records it may contain
ND, the number of deleted records to be replaced before adding new ones

After this, the key array K%( ) is filled in a single line, as is the DE%( ) array two lines later. The reason for cramming these into single lines is to save time in reading it. FOR...NEXT loops contained entirely on a single line work far more quickly than those spanning several lines, especially this far along in the module. The difference is due to not having to scan for line numbers within the loop. As written, the program falls through lines 4770 and 4800 only if there is a DOS error. If it does fall through, the session will be terminated.

Writing the Key File

At the end of each use of the "update" module, Bennett's mail list checks to see if the key file needs to be rewritten to disk. If nothing has happened to change the file in memory, variable UP will equal zero. Any other value triggers a rewrite of the key file. This flag variable is checked in line 1390:

```
1390 IF UP THEN GOSUB 5060:
     REM WRITE KEY FILE
```

Due to a shortage of space on the 4040 diskette, "update" does not use the best method of replacing the "index" file. Ideally, we would first rename the existing file as "old", then save the updated file under the correct name, and finish by scratching "old" after the new copy is properly closed. Unfortunately, this requires enough spare storage on the diskette to hold two copies of the "index" file, and we don't have that much room to spare when the file is full. The same problem keeps us from using the "@" SAVE-WITH-REPLACEMENT option of CBM DOS. It works much the same way, making a spare copy of the new version

before destroying the old one. If you have a larger capacity disk drive, either of the above methods would work well. But on the 4040, we do it this way:

```
5050 REM WRITE OUT KEY FILE
5060 PRINTTAB(11)"WRITING KEY
     FILE"
5070 SCRATCH "INDEX",D(DD) ON
     U(UN)
5080 IF DS > 1 THEN 1690
```

In BASIC 2, substitute:

```
5065 OPEN 15,UN,15
5070 PRINT#15,"SCRATCH" + DD$ +
     "INDEX"
5075 GOSUB 60020:REM CHECK DISK
     STATUS WITHOUT REOPENING
     FILE 15
```

(Considering how often BASIC 2 users will be opening and closing file 15, it would probably be better to open it once in the setup module, and not close it again until the program ends.)

After eliminating the old copy of the file, "update" now proceeds to save a new version:

```
5110 DOPEN#9,D(DD),"INDEX",
     W ON U(UN)
5120 IF DS THEN 1690
5130 PRINT#9,F$C$NR;C$NV;C$ND
5140 IF NV=0 THEN 5170
5150 FOR I=1 TO NV:PRINT#9,
     MID$(STR$(K%(I),2):NEXT
5170 IF ND=0 THEN 5190
5180 FOR I=1 TO ND:PRINT#9,
     DE%(I):NEXT
5190 IF DS THEN 1690
5200 DCLOSE 9
5210 IF DS THEN 1690
```

In BASIC 2, substitute:

```
5110 OPEN 9,UN,9,DD$ + "INDEX,
     SEQUENTIAL,WRITE"
5130 PRINT#9,F$C$NR;C$NV;
     C$ND;C$;
5150 FOR I=1 TO NV:PRINT#9,
     MID$(STR$(K%(I),2)C$;:NEXT
5180 FOR I=1 TO ND:PRINT#9,
     DE%(I)C$;:NEXT
5200 CLOSE 9
```

Notice the use of C$ (containing the carriage return character) as a delimiter in line 5130. This is the only safe way to write multiple variables in a single PRINT# statement. Due to a bug in BASIC 2, it is also the only way to finish a PRINT# statement to the disk, as shown in the alternate lines above. Note too that semicolon spacing between variables is the default, and only needs to be made explicit when variable names could be confused.

The complex expression in line 5150, MID$(STR$(K%(I),2), saves disk storage space at the cost of file rewrite speed. When a numeric variable is PRINT#ed to the disk, a leading space is left for its sign, if any. Since a file can't have a negative record number, the space is wasted — and would fill four disk sectors in Bennett's mail list. The expression strips off the sign space by turning the number into a string and ignoring its first character. If your disk has more capacity, improve your rewrite speed by using a simple PRINT# instead, as in line 5180. (Since the number of deleted records is usually small, little would be gained by using a complex expression in line 5180.)

Also note that once the file is successfully opened, the disk status is not checked again until an attempt has been made to write all the data in the file. This saves time, and if there is an error anywhere in the process, the final check will catch it. DOS errors during PRINT# are not fatal to the program if ignored. However, don't try this when reading the file. Failing to check disk status after every INPUT# will likely halt the program on any DOS error.

What About Alternate Keys?

Careful study of Bennett's mail list will reveal an alternate key, in addition to the primary one, handled in much the same way. We left it out of our discussion because it is not fully developed. At present, it merely records record numbers — a chore easily handled without a second key. In an incomplete 8050 version of the mail list, Bennett went further, using the code field as the alternate key field.

If you need one or more alternate keys, just maintain them along with the primary one, changing all of them whenever any one is altered. Essentially, extra keys are a trade-off. Maintaining them increases the time needed to add, delete, and change records, and adds to the complexity of the program. But maintaining these keys eliminates the delay of sorting before printing records out in ZIP code or other new orders.

Next month we finally get this beast on the road, reading and writing relative file data.

The author may be contacted at 1280 Richland Av., Lincoln, IL 62656.

BEEP! LOST YOUR PROGRAM?

BUS RIDER
LOGIC ANALYZER FOR THE APPLE II

The Bus Rider is a self diagnostic development tool that allows real time analysis of software and hardware in the Apple II computer.

The Bus Rider provides:

- Monitors and saves 512 cycles of the address and data bus, NMI, IRQ, DMA, R/W and 4 external lines.
- Pretrigger viewing of up to 512 samples.
- 4 external inputs with variable threshold reference.
- Display cycle by cycle execution or 6502 disassembled code.

Bus Rider - Disassembled Display

The Bus Rider comes complete with Bus Rider circuit card, reference manual, Bus Rider software diskette, and 10 easy hook external input cable. The total system price is **$395.00**

MC

RC Electronics Inc.
5386 Hollister Avenue, #D
Santa Barbara, CA 93111
(805) 968-6614 • TELEX 295281

Visa

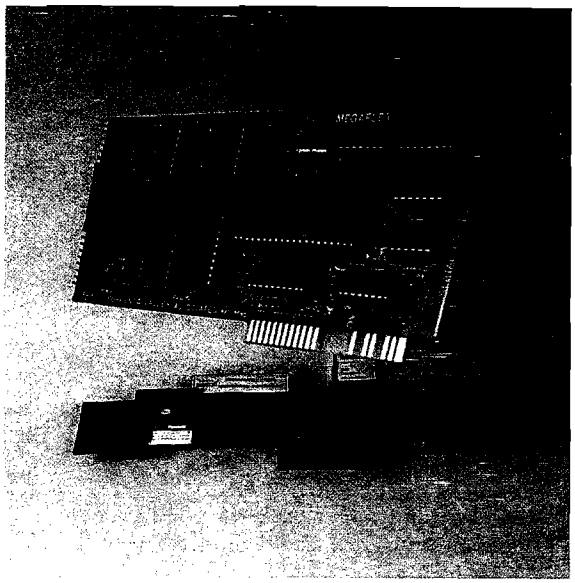
MEGAFLEX® ABILITY

You Pick The Disk System, MegaFlex Controls It!

WITH SOFTDRIVERS FOR A FLEXIBLE FUTURE!

MEGAFLEX—a universal floppy disk controller and modern alternative to the Apple drive system offering increased storage, improved reliability and . . . FLEXIBILITY.

Enjoy megabytes of online storage with your choice of micro, mini, or maxi drives—or even 6Mb with the Amlyn cartridge pack! Ideal for high-capacity storage now, winchester-disk backup later.



The MEGAFLEX secret is to autoboot softdrivers that match the needs of your drive system. All hardware functions are software-controlled. MEGAFLEX can match new drive capabilities without hardware changes. Drive-dependent ROMs have been eliminated.

APPLE *III*? OF COURSE!!

MEGAFLEX is compatible with BASIC, CP/M, Pascal, VISICALC, SOS and DOS-emulation on the Apple III, Apple II, Franklin Ace and Basis. All language features and operating system commands (LOAD, BRUN, etc.) are standard. If you can operate Apple drives you can operate MEGAFLEX! Your Apple software will run without modification too.

BRIDGE THE APPLE FORMAT BARRIER!

The MEGAFLEX diskette does what Apple's cannot—read and write diskettes from other computers! Software-controlled industry-standard IBM 3740 or System 34 type formats allow the MEGAFLEX library of reformatting software to read and write Altos, Radio Shack, Osborne, and IBM PC diskettes. (Call for the latest software details.)

MORE STORAGE, MORE UNIVERSAL FEATURES, LOWEST COST

MEGAFLEX with 8" maxi or high density 5.25" minis gives you 1.2 Megabyte of formatted data per diskette for 8 times the file and data size!

MEGAFLEX offers flexible software choices:

- data rate (250/500 Kbits per second),
- single and double density recording, and
- single/double sided drive operation (max 4 drives).

MEGAFLEX has the lowest chip count of any controller today! This means less power, a cooler Apple and better reliability.

Lowest price, highest performance, that's MEGAFLEX!™

MEGABYTER
A Division of SVA

11722 SORRENTO VALLEY ROAD
SAN DIEGO, CA
(619) 452-0101
TWX 910-335-2047 APPLE TWO SDG

TRADEMARKS CP/M—Digital Research

A Binary Search Routine

by Alfred J. Bruey

This article describes the binary search technique and then presents two demonstration programs. One sets up a sorted test file that can be searched using a binary search; the other sets up a sorted test file and allows a record to be inserted in its proper place. This binary search technique can be used as the basis for a complete data base system.

Sort Routines require:

Any Microsoft BASIC computer

Your computer is handy for keeping lists of names, right? But how do you find a particular name, especially if your list is in random order? You could write a program to start at the beginning of the file and compare items one by one with the name you want. Of course, this process would be extremely time-consuming. And if the name you were searching for wasn't there, you wouldn't know until you had searched the entire list!

Unfortunately, there is no better method to search for an item if the list is in random order. Therefore, you should make sure your file is in alphabetical order; the rest of this discussion will assume your file is in order.

The binary search routine presented here can be used to find items on a sorted list. You can also use this routine to find where you should place a new item.

The principle of the binary search is as follows (note that in any of the steps below, you are done if you get an exact match).

1. Check the first item on the list. If the value you're searching for is less than this, the item isn't on the list.
2. Go to the middle value on the list. If the value there is greater than your search value, you've got the search narrowed to the first half of the list. If not, you've got it narrowed to the last half of the list.
3. Next try the middle value of the interval found in step 2 above. Now you have the search narrowed down to one quarter of the list.
4. The next comparison will narrow the search to 1/8th of the list, the next to 1/16th, etc. Therefore, counting your compare with the first item on the list, you will have your missing value narrowed down to 1/16th of the list after only five compares, 1/32nd of the list after six compares, 1/64th after seven compares, etc.
5. Continue this process until you find the item, or reduce the list to one item.

Subroutine Description

The binary search subroutine shown as part of figure 1 (lines 10000 and greater) performs as follows:

1. The value you are searching for is put in NM$.
2. The file being searched is assumed to be in array N$.
3. If the value NM$ is found in array N$, its location is returned as the value of variable J. (The first item in array N$ is N$(0).)
4. If NM$ isn't found in array N$, the value of J will be the index of the next array value larger than NM$.

Let's look at some examples with a file N$ that contains the following

***Listing 1:* Routine to Find Records**

```
10 REM ***************************
20 REM ROUTINE TO FIND RECORDS*
30 REM ***************************
35 DIM N$(1000):N=10
40 DATA BILL,CARL,CARL,DON,JOHN,MARIE,MARY,PETE,ROGER,ROGER,ZZZZZ
50 FOR I=0 TO N
60 READ N$(I)
70 NEXT I
80 INPUT "ENTER VALUE YOU WANT TO SEARCH FOR (ENTER XXX TO STOP)";NM$
90 IF NM$="XXX" THEN STOP
100 GOSUB 10000
110 PRINT "INDEX ";J;" FOUND FOR SEARCH ARGUMENT ";NM$
120 GOTO 80
10000 REM ***************************
10010 REM BINARY SEARCH SUBROUTINE*
10020 REM ***************************
10030 J=0
10040 IF J=1 THEN 10200
10050 IF N<=0 THEN 10200
10060 IF NM$<N$(0) THEN 10200
10070 J1=0:J2=N
10080 J=INT((J1+J2)/2)
10090 IF NM$=N$(J) THEN 10140
10100 IF NM$<N$(J) THEN J2=J:GOTO 10120
10110 J1=J
10120 IF J<>INT((J1+J2)/2) THEN 10080
10130 J=J+1:GOTO 10200
10140 IFJ=0 THEN 10200
10150 IF J=1 AND NM$=N$(J-1) THEN J=0:GOTO10200
10160 IF J=1 THEN 10200
10170 FOR J=J TO 1 STEP -1
10180 IF NM$<>N$(J) THEN J=J+1:GOTO 10200
10190 NEXT J
10200 RETURN
```

names (we will be using this file later in the sample computer program):

Index No.	Name
0	BILL
1	CARL
2	CARL
3	DON
4	JOHN
5	MARIE
6	MARY
7	PETE
8	ROGER
9	ROGER
10	ZZZZZ

Note the following:

1. The same name can appear more than once on the list. The value of J returned will be the index of the first appearance of the name.
2. The last value in the file must be larger than the last value that could appear. The value ZZZZZ will usually satisfy this requirement.

The program in figure 1 sets up the sample file shown earlier. With this routine, we can enter search values and receive index values from the subroutine.

Enter and run this program for the following search values as shown in the NM$ column. The response should be the index value given in the J column.

NM$	J
JOHN	4
ADAM	0
BILL	0
YOST	10
MARIE	5
NANCY	7
PETE	7

Note that ADAM and BILL both return a value of 0, and NANCY and PETE both return a value of 7. You will have to put a line in your program to check whether you have actually found the value of NM$, or whether you have found the value that would have followed NM$ if it had been on the file.

The next two lines of coding can be added to what you've previously entered. These lines allow you to see if the value was found or not. Enter them and run the program again to check the results.

```
112 IF N$ (J) = NM$ THEN PRINT
"RECORD FOUND":PRINT:PRINT
114 IF N$(J) < > NM$ THEN PRINT
"RECORD NOT FOUND":PRINT:PRINT
```

***Listing 2:* Routine to Insert Records**

```
10 REM *************************
20 REM ROUTINE TO INSERT RECORDS*
30 REM *************************
35 DIM N$(1000):N=10
40 DATA BILL,CARL,CARL,DON,JOHN,MARIE,MARY,PETE,ROGER,ROGER,ZZZZZ
50 FOR I=0 TO N
60 READ N$(I)
70 NEXT I
80 INPUT "ENTER VALUE YOU WANT TO INSERT (ENTER XXX TO STOP)";NM$
90 IF NM$="XXX" THEN 250
100 GOSUB 10000
120 REM *************************
130 REM ROUTINE TO INSERT RECORD *
140 REM *************************
150 PRINT"INSERTING RECORD IN FILE"
160 FOR I=N TO J STEP -1
170 N$(I+1)=N$(I)
180 NEXT I
190 N=N+1
200 N$(J)=NM$
210 PRINT "RECORD INSERTION COMPLETE"
220 PRINT "*************************
230 PRINT:PRINT
240 GOTO80
250 REM PRINT NEW FILE
260 FOR I=0 TO N
270 PRINT N$(I)
280 NEXT I
290 STOP
10000 REM *************************
10010 REM BINARY SEARCH SUBROUTINE*
10020 REM *************************
10030 J=0
10040 IF J=1 THEN 10200
10050 IF N<=0 THEN 10200
10060 IF NM$<N$(0) THEN 10200
10070 J1=0:J2=N
10080 J=INT((J1+J2)/2)
10090 IF NM$=N$(J) THEN 10140
10100 IF NM$<N$(J) THEN J2=J:GOTO 10120
10110 J1=J
10120 IF J<>INT((J1+J2)/2) THEN 10080
10130 J=J+1:GOTO 10200
10140 IFJ=0 THEN 10200
10150 IF J=1 AND NM$=N$(J-1) THEN J=0:GOTO10200
10160 IF J=1 THEN 10200
10170 FOR J=J TO 1 STEP -1
10180 IF NM$<>N$(J) THEN J=J+1:GOTO 10200
10190 NEXT J
10200 RETURN
```

Now let's see how to insert new records into a sorted list. Figure 2 shows the coding that, along with the search subroutine, will perform this function. This program will ask you to enter values. As you enter them, they will be placed in their proper (alphabetical) place in the file. When you enter 'XXX' as a record, the program will stop and print out your new list. You will lose all your new values when the program ends, but this isn't serious since this is only a test program. In practice, you would read all the data in from a tape or disk and then write them back to a tape or disk when all the changes have been made.

For an exercise, write a delete routine. You should be able to model it after the insert routine in figure 2. Notice that you don't always have to check for the full record. You can, for example, replace N$(I) with LEFT$ (N$(I),3) in the subroutine if you only want to check the first three characters for a match.

As you might have guessed, the binary search routine can be set up as the foundation for a complete file maintenance system. If you are able to find a record, it's usually a simple matter to change it or delete it. The major disadvantage is that you must be able to hold your entire file in RAM (Random Access Memory) for the length of the run. This restricts the size of the file you can use. As an extension, you might want to write a routine to read in, say, 100 records and search through them, then read in another 100 records and search through them, etc. This method would be slower, but it would allow you to search disk or tape files of any length.

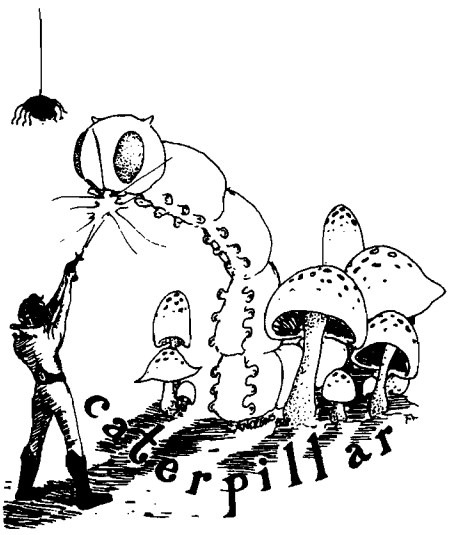
Contact Mr. Bruey at 201 S. Grinnell St., Jackson, MI 49203.

MICRO

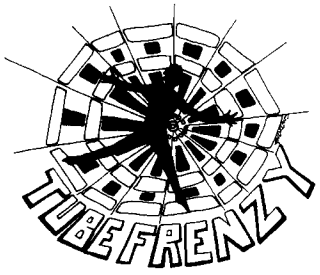
AARDVARK

TRS-80 COLOR OSI VIC-64 VIC-20 SINCLAIR TIMEX

QUEST – A NEW IDEA IN ADVENTURE GAMES! Different from all the others. Quest is played on a computer generated map of Alesia. Your job is to gather men and supplies by combat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time. 16k TRS-80, TRS-80 Color, and Sinclair. 13K VIC-20. $14.95 each.



CATERPILLAR
O.K., the Caterpillar does look a lot like a Centipede. We have spiders, falling fleas, monsters traipsing across the screen, poison mushrooms, and a lot of other familiar stuff. COLOR 80 requires 16k and Joysticks. This is Edson's best game to date. $19.95 for TRS 80 COLOR.



TUBE FRENZY
(by Dave Edson)
This is an almost indescribably fast action arcade game. It has fast action, an all new concept in play, simple rules, and 63 levels of difficulty. All machine code, requires Joysticks. Another great game by Dave Edson. TRS 80 COLOR ONLY. 16k and Joysticks required. $19.95.

ADVENTURES!!!
These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch.")

Adventures require 16k on TRS80, TRS80 color, and Sinclair. They require 8k on OSI and 13k on Vic-20. Derelict takes 12k on OSI. $14.95 each.

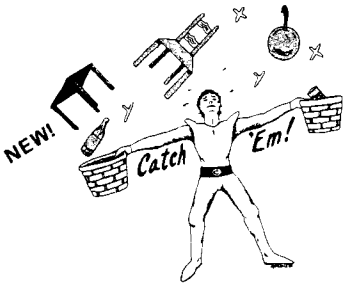
PROGRAMMERS!
SEE YOUR PROGRAM IN THIS SPACE!!
Aardvark traditionally pays the highest commissions in the industry and gives programs the widest possible coverage. Quality is the keyword. If your program is good and you want it presented by the best, send it to Aardvark.

ESCAPE FROM MARS
(by Rodger Olsen)
This ADVENTURE takes place on the RED PLANET. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)
This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

HAUNTED HOUSE (by Bob Anderson)
It's a real adventure—with ghosts and ghouls and goblins and treasures and problems – but it is for kids. Designed for the 8 to 12 year old population and those who haven't tried Adventure before and want to start out real easy.

DERELICT
(by Rodger Olsen & Bob Anderson)
New winner in the toughest adventure from Aardvark sweepstakes. This one takes place on an alien ship that has been deserted for a thousand years – and is still dangerous!



CATCH'EM
(by Dave Edson)
One of our simplest, fastest, funnest, all machine code arcade games. Raindrops and an incredibe variety of other things come falling down on your head. Use the Joysticks to Catch'em. It's a BALL! – and a flying saucer! – and a Flying YI– and so on. TRS 80 COLOR. $19.95.

BASIC THAT ZOOOMMS!!
AT LAST AN AFFORDABLE COMPILER!
The compiler allows you to write your programs in easy BASIC and then automatically generates a machine code equivalent that runs 50 to 150 times faster.

It does have some limitations. It takes at least 8k of RAM to run the compiler and it does only support a subset of BASIC– about 20 commands including FOR, NEXT, END, GOSUB, GOTO, IF, THEN, RETURN, END, PRINT, STOP, USR (X), PEEK, POKE, *, /, +, -, >, <, =, VARIABLE NAMES A-Z, SUBSCRIPTED VARIABLES, and INTEGER NUMBERS FORM 0-64K.

TINY COMPILER is written in BASIC. It generates native, relocatable 6502 or 6809 code. It comes with a 20-page manual and can be modified or augmented by the user. $24.95 on tape or disk for OSI, TRS-80 Color, or VIC.

Please specify system on all orders

ALSO FROM AARDVARK – This is only a partial list of what we carry. We have a lot of other games (particularly for the TRS-80 Color and OSI), business programs, blank tapes and disks and hardware. Send $1.00 for our complete catalog.

AARDVARK - 80
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

Phone Orders Accepted 8:00 a.m. to 4:00 p.m. EST. Mon.-Fri.

BASIC Renumber for OSI

by Paul Krieger

A BASIC language routine to renumber BASIC programs in memory and save to tape.

Renumber
requires:
OSI C1P

This BASIC language routine is written for an OSI C1P. It will probably run on other OSI machines and, I suspect, could be adapted to most Microsoft BASIC machines as well. To do this you would have to: 1. change X=769 throughout the program to the appropriate starting address for your machine, 2. revise the instruction table lines 40320-40490 for your particular BASIC, and 3. change the syntax of the BASIC to your format.

To operate this program, first key it into your machine after a cold start and then test it to make sure there are no keyboard errors or omitted lines. You can test it by typing RUN40280. The first message on the screen will be "ENTER RANGE OF OLD #'s YOU WISH TO UPDATE(LOW,HIGH)". Enter the lower number, comma, higher number. For the test enter 40280,41500, which is the range of this routine. The program will then ask you for the new number (BEGIN,STEP). Enter the new beginning number and the increment you want between numbers. An appropriate answer for this would be 1,1, which will cause the first line to be 1 instead of 40280, and each line thereafter will be 1 greater. There will be a pause after your reply while the program builds the branch table.

Once the preliminary work is done, the progam will type the message "START TAPE RECORDER IN RECORD/PLAY MODE NOW". This indicates it is ready to make the updated copy of the program. Start your recorder using a blank tape and type space, return. You will see the program listing with the new numbers. After the tape is made, you will have the unchanged program in memory and a copy of it with new line numbers on tape. To run the new version press BREAK and cold start your machine. Then LOAD the tape as you would any other program. To renumber any other BASIC program, first load it into memory then load this renumber routine in with it by mounting the cassette and typing load again.

This program relies on the cassette tape as a working medium. There are two reasons why I chose to write the program this way. First, by making a tape instead of changing line numbers in memory, it is easy to handle line

***Listing 1:* Renumber BASIC**

```
40280 REM ROUTINE TO RENUMBER BASIC BY Paul Krieger
40290 REM          Nov. 8, 1981
40300 REM BT=BRANCH TABLE, LN=LINE NUMBER OF DESTINATION
40310 DIMIT$(67):DIMBT(200):DIMLN(200)
40320 IT$(0)="END":IT$(1)="FOR":IT$(2)="NEXT":IT$(3)="DATA"
40330 IT$(4)="INPUT":IT$(5)="DIM":IT$(6)="READ":IT$(7)="LET"
40340 IT$(8)="GOTO":IT$(9)="RUN":IT$(10)="IF":IT$(11)="RESTORE"
40350 IT$(12)="GOSUB":IT$(13)="RETURN":IT$(14)="REM":IT$(15)="STOP"
40360 IT$(16)="ON":IT$(17)="NULL":IT$(18)="WAIT":IT$(19)="LOAD"
40370 IT$(20)="SAVE":IT$(21)="DEF":IT$(22)="POKE":IT$(23)="PRINT"
40380 IT$(24)="CONT":IT$(25)="LIST":IT$(26)="CLEAR":IT$(27)="NEW"
40390 IT$(28)="TAB":IT$(29)="TO":IT$(30)="FN":IT$(31)="SPC("
40400 IT$(32)="THEN":IT$(33)="NOT":IT$(34)="STEP":IT$(35)="+"
40410 IT$(36)="-":IT$(37)="*":IT$(38)="/":IT$(39)="^"
40420 IT$(40)="AND":IT$(41)="OR":IT$(42)=">":IT$(43)="="
40430 IT$(44)="<":IT$(45)="SGN":IT$(46)="INT":IT$(47)="ABS"
40440 IT$(48)="USR":IT$(49)="FRE":IT$(50)="POS":IT$(51)="SQR"
40450 IT$(52)="RND":IT$(53)="LOG":IT$(54)="EXP":IT$(55)="COS"
40460 IT$(56)="SIN":IT$(57)="TAN":IT$(58)="ATN":IT$(59)="PEEK"
40470 IT$(60)="LEN":IT$(61)="STR$":IT$(62)="VAL":IT$(63)="ASC"
40480 IT$(64)="CHR$":IT$(65)="LEFT$":IT$(66)="RIGHT$"
40490 IT$(67)="MID$"
40500 REM     INITIALIZE
40510 PRINT "ENTER RANGE OF OLD #'S YOU WISH TO UPDATE"
40520 INPUT"(LOW,HIGH)";LR,HR
40530 INPUT"ENTER NEW NUMBER: (BEGIN,STEP)";BG,ST
40540 X=769:REM BEGIN OLD DESTINATION PASS "ODP"
40550 Q=0:REM SET TABLE INDEX TO BEGINNING
40560 NA=PEEK(X+1):REM GET NEXT INSTRUCTION HIGH BYTE
40570 NA=NA*256:REM SETUP HIGH
40580 NB=PEEK(X):NA=NA+NB:REM ADD IN LOW
40590 REM NA IS NEXT ADDRESS, NB IS WORK
40600 IFNA=0THEN40980:REM EXIT OLD DEST PASS
40610 L=PEEK(X+3):L=L*256:NB=PEEK(X+2):L=L+NB
40620 IFL>HRTHEN40980
40630 W=X+4:REM TO TEXT OF INST
40640 FORI=0TO72:REM SCAN FOR BR
40650 C=PEEK(W+I):REM BRANCH TABLE ARGUMENT
40660 REM 144=ON, 44=COMMA AFTER "ON"
40670 REM 136=GOTO,140=GOSUB,160=THEN,137=RUN
40680 IFC=144THENS=1:REM "ON" SET SWITCH TO 1
40690 IFS=1THEN40720
40700 IFC=136ORC=140ORC=160ORC=137THENGOSUB40770
```

***Listing 1:* Renumber BASIC** *(continued)*

```
40710 GOTO40730
40720 IFC=136ORC=140ORC=160ORC=137ORC=44THENGOSUB40770
40730 IFC=0THENI=72:S=0:REM RESET "ON-GOTO" SWITCH
40740 NEXTI
40750 X=NA:GOTO40560:REM CONTINUE W NEXT LINE
40760 REM     ENTER HERE FOR BRANCHING ONLY
40770 REM I NOW POINTS AT POSSIBLE "THEN"
40780 IFC<>160THEN40800
40790 B=PEEK(I+W+1):IFB>57THENRETURN:REM "THEN" W EXPR=IGNORE
40800 B=C
40810 FORK=0TO6:REM FIND LEFTMOST # OF DESTINATION
40820 C=PEEK(K+I+W)
40830 IFC>47ANDC<58THENK=K+6
40840 NEXTK
40850 KR=K-6:REM FIND RIGHT SIDE
40860 FORK=KRTOKR+6:C=PEEK(K+I+W)
40870 IFC<48ORC>57THENK=K+6
40880 NEXTK
40890 K=K-8
40900 J=10
40910 BT(Q)=PEEK(K+I+W):BT(Q)=BT(Q)-48
40920 K=K-1
40930 C=PEEK(K+I+W)
40940 IFC<48ORC>57THEN40970
40950 C=C-48:C=C*J:J=J*10:BT(Q)=BT(Q)+C
40960 GOTO40920
40970 C=B:Q=Q+1:RETURN
40980 REM     LINK TO DESTINATION
40990 X=769
41000 BD=BG
41010 NA=PEEK(X+1):NA=NA*256:NB=PEEK(X):NA=NA+NB
41020 IFNA=0THEN41130
41030 L=PEEK(X+3):L=L*256:NB=PEEK(X+2):L=L+NB
41040 IFL>HRTHEN41130
41050 FORQ=0TO200:REM LINK OLD TO NEW
41060 Y=X+3:LN=PEEK(Y):LN=LN*256:REM HIGH OF LINE #
41070 Y=X+2:LO=PEEK(Y):LN=LN+LO:PRINTLN;:REM PRINT LINE #
41080 NEXTQ
41090 IFL<LRORL>HRTHEN41110
41100 BD=BD+ST
41110 X=NA:GOTO41010
41120 REM       RENUMBER
41130 SAVE:PRINT"START TAPE RECORDER IN RECORD/PLAY MODE NOW"
41140 INPUT"PRESS SPACE/RETURN WHEN PAST LEADER";B$
41150 Q=0
41160 X=769
41170 NA=PEEK(X+1):NA=NA*256:NB=PEEK(X):NA=NA+NB
41180 IFNA=0THEN41500
41190 Y=X+3:LN=PEEK(Y):LN=LN*256:REM HIGH OF LINE #
41200 Y=X+2:LO=PEEK(Y):LN=LN+LO:IFLN>HRTHEN41500
41210 IFLN<LRORLN>HRTHEN41240
41220 PRINTBG;:BG=BG+ST
41230 GOTO41250
41240 PRINTLN;:REM PRINT LINE#
41250 W=X+4
41260 FORI=0TO72
41270 C=PEEK(W+I)
41280 IFC=0THENX=NA:I=73:GOTO41430
41290 IFC=44THENPRINTCHR$(C);:GOTO41340
41300 IFC<128ORC>195THEN41420
41310 C=C-128:PRINTIT$(C);
41320 IFLN<LRORLN>HRTHEN41430
41330 C=C+128:IFC=160THENB=PEEK(W+I+1):IFB>57THEN41430
41340 IFC=144THENS=1:REM SET "ON-GOTO" SWITCH
41350 IFS=1THEN41380
41360 IFC=136ORC=140ORC=160ORC=137THEN41400
41370 GOTO41430
41380 IFC=136ORC=140ORC=160ORC=137ORC=44THEN41400
41390 GOTO41430
41400 GOSUB41460
41410 I=I+B-1:GOTO41430:REM BYPASS NEW DEST #
41420 PRINTCHR$(C);
41430 NEXTI
41440 S=0:PRINT
41450 GOTO41170
41460 B$=STR$(LN(Q)):B=LEN(B$):PRINTRIGHT$(B$,B-1);
41470 B=LEN(STR$(BT(Q))):REM PRINTING NEW LINE #
41480 Q=Q+1
41490 RETURN
41500 FORDL=1TO4000:NEXTDL:POKE517,0:END
```

numbers that can vary from one digit to as many as five. Second, by making a tape, the program can renumber itself, which it could not do if it were changing memory. A byproduct is that a copy of the original program is still in memory in case there are bugs in this renumbering routine.

BASIC is kept in memory in a compressed format. The first two bytes are the address of the next instruction in memory. This is a binary value with a range of from 771 to 65383. The next two bytes are the current BASIC line number, which is also in binary. The range of this field is 0 to 63999. Then comes the text of the BASIC instruction. Any BASIC command, statement, operator, or function is reduced to a single character in the range 128 (\$80) to 195 (\$C3).

A line number destination (branch), as in a GOTO, is kept in the graphic format. To renumber in memory it is necessary to deal with the branch line numbers by expanding or compressing each line in memory from the end of the subject line to the end of the BASIC instructions.

The fourth line of the program includes two dimension statements using 200. This represents the maximum number of branches in a program. So far this has worked fine for me, but you may wish to make this value larger or smaller according to the size of your programs or memory or both.

If you load this routine into memory and all the line numbers are larger than those already in memory, there probably won't be a timing problem. If the line numbers of this routine are smaller than those already in memory, there may be a timing problem as the machine will relocate all the instructions in memory once for each line read in. If this is the case, you will see partial lines being read from the tape and a lot of error messages on the screen. You can overcome this problem by placing some nulls in the renumber program prior to creating the master tape.

The author may be contacted at 3268 S. Cathay Cr., Aurora, CO 80013.

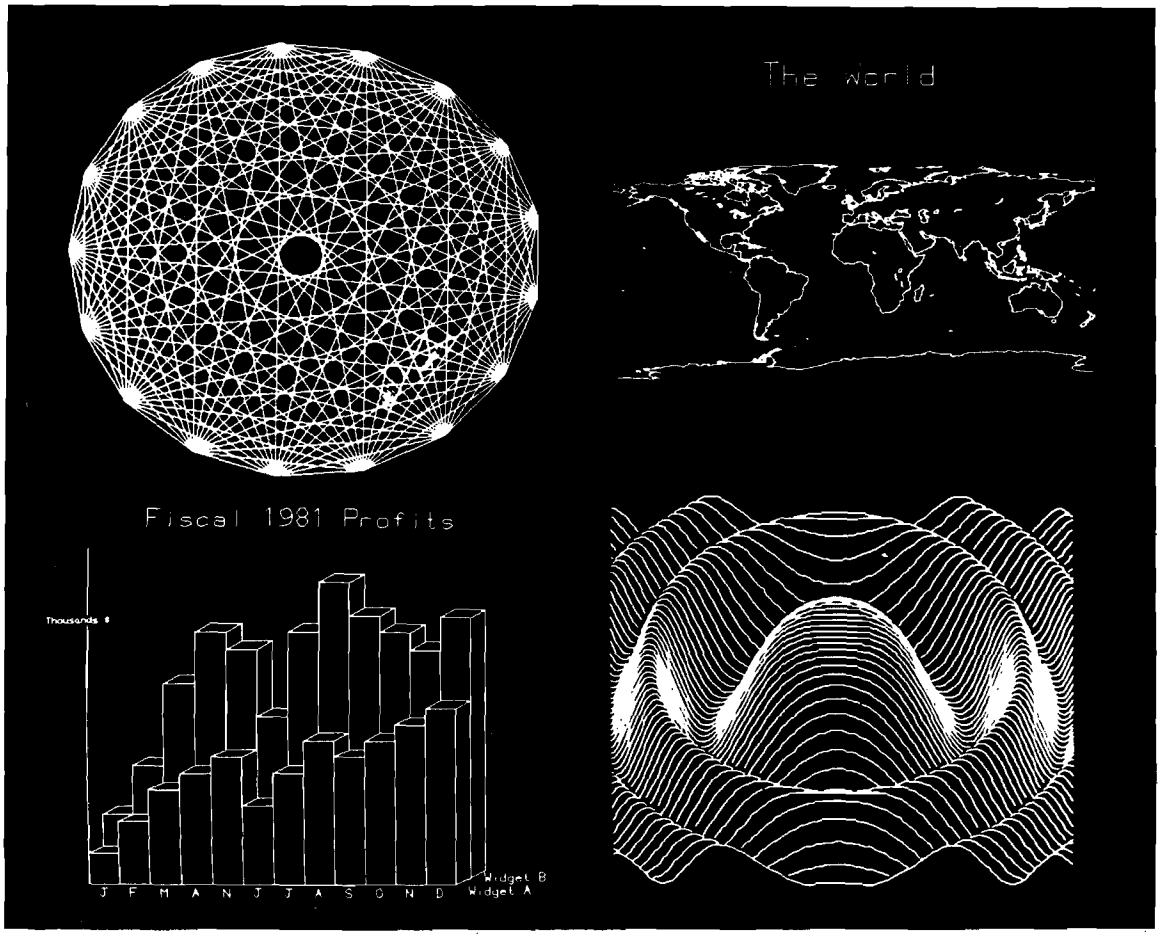
MICRO

ANNOUNCING ElectroScreen™

only $595

the Superior Alternative

to the Traditional Alphanumeric Terminals



The ElectroScreen™ Intelligent Graphics Board Features:

Graphics

- 512 x 480 resolution bit-mapped display
- Interleaved memory access — fast, snow-free updates

Intelligence

- 6809 on-board mpu
- 6K on-board firmware
- STD syntax high level graphics command set
- Removes host graphics software burden
- Flexible text and graphics integration
- Multiple character sizes
- User programs can be run on-board

Terminal

- Terminal emulation on power-up
- 83 characters by 48 lines display
- Easy switching among user-defined character sets
- Fast hardware scrolling

Additional Features

- SS-50C and SS-64 compatible board
- Board communicates with host through parallel latches
- Composite and TTL level video output
- 8 channel 8 bit A/D converter
- Board occupies 4 address bytes

See your dealer today!

The ElectroScreen manual is available for $10, credited toward purchase of the board.

The ElectroScreen has a 90 day warranty from purchase date.

Dealers, please contact us for our special introductory package.

Privac Inc (703) 671-3900

3711 S. George Mason Dr., Falls Church, Va. 22041

SuperPET APL

by Terry Petersen

This article gives a brief description of several APL textbooks currently available, an overview of APL in general, and the SuperPET's microAPL in particular.

Before I owned a Commodore SuperPET, I had heard of a programming language called APL but never had access to a machine that could run it. Therefore, my curiosity about this untried language influenced my decision to buy a SuperPET. I think it is fair to say microAPL is the most unorthodox of the languages supplied with the SuperPET. Programmers raised on FORTRAN and BASIC refer to it as "the closest to hieroglyphics I've ever seen," and "a write-only language." MicroAPL is also the largest of the four interpreters by Waterloo Computer Systems, Ltd. (WCS) that came with my SuperPET (it nearly fills the 64K bank-switched expansion RAM). Conversely, it has the smallest user manual — 108 pages *versus* 137 for microPascal and 221 for microBASIC. I soon concluded I needed more help getting started in APL than was available in the Waterloo documentation. To learn more about the language I pestered my local librarian to obtain several APL textbooks *via* interlibrary loan.

Books on APL

APL inventor Kenneth Iverson wrote the first book about the language. Published in 1962, it is entitled *A Programming Language*.[1] It describes some of the rationale of APL and how to implement it. I recommend Mr. Iverson's book *only* to those interested in APL's guts — it is not a good tutorial for learning to write APL programs.

Books in Print currently lists about a dozen APL textbooks. I borrowed five of them, selected more or less at random, all of which are better than the original for learning APL programming. I found the most folksy approach to teaching APL in Howard A. Peelle's *APL: An Introduction*.[2] Written in an unusual style, this book is organized into nine tutorial sessions. The pages are printed to look like an APL terminal printout with hand-written notes added. The lessons seem easy to follow and there are frequent questions for the reader to test his understanding of the material. I think this book is a good introduction to APL for those with little or no computer experience. Others will probably find it a bit plodding.

APL: An Interactive Approach by Leonard Gilman and Allen J. Rose[3] also assumes no particular computer or mathematics background. The book is geared to IBM equipment, as are all the others I've seen, but it is fairly complete in describing the language elements, and is quite usable with the SuperPET as long as you ignore the information about workspace storage, etc. (Note: In APL you save and load workspaces rather than programs.) This book seems to be more widely available than Peelle's; I found it in a university bookstore.

Of the five textbooks I selected, my favorite is *Handbook of APL Programming* by Clark Wiedmann.[4] This book is more terse than Iverson's or Rose's, but is more explanatory than WCS's manual. The copy I borrowed from the library was clothbound, but the copyright information page indicates it is available in paperback also.

APL Programming and Computer Techniques by Harry Katzan, Jr.[5] is notable for its inclusion of several real APL programs for study. The edition I saw, however, was published in 1970 and appears rather dated; for example, there is no mention of the domino function. Perhaps later editions are more up-to-date.

Finally, I looked at *A Course in APL with Applications* by Louis D. Grey.[6] This book seems to me to have been hastily prepared, with many (typographical?) errors as well as what appear to be outright program errors. Such confusion does not contribute positively to the learning experience and is most unwelcome in a textbook.

Accessible Programming

When Kenneth Iverson invented APL, FORTRAN was practically the only high-level language available for scientific programming. At that time FORTRAN dialects were very restrictive and demanded that the programmer be a fairly sophisticated computerist. Even if you could get your program to compile successfully, there was a good chance some esoteric feature of the compiler's method of internal number representation would give unexpected results.

The main intent of Iverson's new language was to make programming more accessible by freeing programmers (still presumed to be mathematically oriented) from mundane considerations, such as whether or not a number is stored in the computer in integer or floating-point form, or whether or not a particular variable is a scalar or an array of dimension x. APL achieves this intent; the same APL variable may contain (at different times during program execution) integer, floating-point, or even character data, and it may become a scalar or an array merely by assigning such data to it. A very powerful side-effect of this lack of 'type' is that an APL 'function' may return *with no difference in coding* scalar, vector, or array results, depending only on given argument(s)!

Compared to other languages of the

early sixties, APL is certainly a paragon of versatility. Even today, I think it is unmatched in its freedom from variable types. This freedom comes at a price, of course. Since there are no declarations of variable types (as in Pascal) nor implicit types (as in FORTRAN and BASIC), you are forced to discover from its context what kind(s) of data a variable contains. This process of discovery can be puzzling and time-consuming when you read an APL program written by someone else — or yourself, six months earlier.

Also, since APL tends to substitute array operations for things that would be written as loops in other languages, its coding is unusually compact. I doubt that even the most experienced APL programmers skim through unfamiliar APL code the way you might with Pascal or well-written BASIC. However, APL's compactness does have its virtue: microAPL is the SuperPET's fastest interpreter, hands-down. There simply isn't as much source code to scan in performing a given task as in other languages. For example, consider as a benchmark the filling of an array with the sequence 1, 2, 3, ..., 1000. This task takes (as coded below) 11 seconds in microBASIC, 5.25 seconds in CBM BASIC, and only 1.18 seconds in microAPL.

APL Implementations

APL implementations, including microAPL, contain many more built-in functions than are found in most other languages, even on mainframe computers. There are functions for finding the maximum or minimum value in an array, sorting arrays, and 'cutting' and 'pasting' arrays to make smaller or larger arrays. There is even a function, called the 'domino,' for finding the 'least-squares' fit of data to a model equation! (Domino will also, trivially, invert a matrix.)

Aside from being rather difficult to read, APL's worst deficiency, in my estimation, is its primitive branching mechanism. Its only branch instruction is a close relative of BASIC's 'ON X GOTO 1,...,N.' There is no IF... THEN...ELSE, WHILE..., UNTIL..., etc. — in short, no *structured* programming. This is not as bad as it sounds because APL's rich complement of built-in functions and extensive use of arrays obviate many loops and branches. However, since it lacks the sorts of program-flow control statements found in most other, more modern, languages, efficient APL coding requires a different programming style and mindset. For example, the array-filling benchmark mentioned above is coded in BASIC as follows:

```
100 DIM A(1000)
110 FOR I = 1 TO 1000
120 A(I) = I
130 NEXT I
```

In APL it is written as:

```
A ← 1000 ⍴ ⍳ 1000
```

where the meaning of the above one-liner is "Assign to A the 1000-element vector formed from the integers 1 to 1000." It's not too hard to see why the APL interpreter makes such short work of this benchmark. It scans one short line of source while the poor BASIC interpreter is stuck with scanning lines 120 and 130 a thousand times!

As an example of how *not* to write APL, you could code this benchmark more closely to the BASIC version this way:

```
   A ← 1000 ⍴ 0
   I ← 1
XX: A[I] ← I
   I ← I + 1
   → (I < 1001) / XX
```

where the meaning of these APL lines may be guessed by comparison with the BASIC version. This awful mess takes 121 seconds to run and is a glaring example of what might be called 'pidgin APL.' I hope my remark about mindset is now clear.

I should hasten to add, for the benefit of any fervent structured-programming enthusiasts, that it is possible to impose some of the cosmetics of structured programming on APL in order to make it more readable. If you have a SuperPET, see the excellent work done by an anonymous WCS programmer in the sample workspace "MASTERMIND" on the SuperPET tutorial diskette. This sort of veneer, however, doesn't really make up for the lack of control statements in APL.

A feature APL does share with modern structured languages is the high degree of modularity. APL programs usually are written as a collection of functions, similar to Pascal procedures and functions, which may use either local or global variables. Each function may have zero, one, or two arguments, and may or may not return an explicit result. The limitation to two arguments is not as restrictive as you might imagine because each argument can be an array. However, any assumption within the function about the rank of its arguments reduces its generality. For example, consider the following APL function:

```
[0] Z ← EQUALS 3 A
[1] Z ← A=3
```

If A is a scalar, the function EQUALS 3 returns a scalar value of 1 if A is 3 and returns 0 otherwise. In addition, if A is any numeric array whatever, this function will return an *array* of 1's and 0's of the same shape as A. On the other hand, the following function will accept only one type of argument — a numeric vector (and it ignores all but the first three elements):

```
[0] VOLUME ← PARALLELAPIPED X

[1] VOLUME ← X[1] × X[2] × X[3]
```

As you might suspect, this is another example of pidgin APL. A more proper APL version of PARALLELAPIPED would be:

```
[1] VOLUME ← ×/ X
```

This second version uses the 'reduction' (/) operator to signify multiple application of the X's operator. In this version a vector argument would result in the scalar product of all the elements of the vector; and an array argument would yield an array of one lower rank (one less subscript) with each element

equal to the product of the elements of the argument array obtained by fixing all but the last subscript! (Mindset again.)

APL's extensive use of arrays tends to require large amounts of memory. On the SuperPET this is particularly troublesome because the microAPL interpreter stores all numeric quantities in floating-point form, requiring five bytes each. Furthermore, microAPL doesn't allow all the available RAM to be used for variable storage. I haven't worked out just how the division between program and variable space is decided, but it works out such that an otherwise empty workspace cannot contain even one 3000-element numeric array. However, it can contain *four* 1000-element arrays! In most cases, it seems you probably will run out of variable space before the user RAM is actually full. The RAM-gobbling situation could be reduced dramatically if some form of byte- or integer- (double-byte) numeric storage could be used where appropriate.

APL Character Set

APL has a strange character set. The wealth of built-in functions of APL are invoked *via* a vast array of unusual characters. I find this a problem even though I'm familiar with the Greek alphabet; those who are not face even greater difficulty. The situation in APL could be compared to requiring BASIC programmers to read and write their programs in the 'tokens' used internally by the interpreter instead of English keywords. It's a pity Mr. Iverson didn't invent the keyword-token translator along with the rest of APL.

Besides requiring memorization, the characters of APL present a practical problem when printing and typing them. There are special APL printwheels available for daisy-wheel printers, and some dot-matrix printers may be programmed to 'draw' APL characters; but the APL keyboard is still troublesome. There are so many APL characters they won't fit all at once on a regular keyboard, so some of them are formed by 'overstriking' to make one character out of two simpler characters. This is awkward and requires you to remember which two characters to use. Mercifully the two characters may be typed in either order. (Ed. note: Terry had to limit his program examples to ones containing the Greek and math characters our typesetter has.)

In summary, there are things I like very much about APL. Its free-form variables permit very elegant and clever coding of mathematical problems. (I've written a one-line function that computes the next generation of a LIFE game, but I can fill only about half the screen because of the SuperPET's RAM limitation.) There are also things I dislike about APL. Its odd character set and lack of flow-control statements are anachronisms. MicroAPL, specifically, seems to be a pretty faithful implementation of the IBM language, as advertised. It could benefit significantly, however, from some micro-oriented extensions, such as integer- or byte-numeric arrays (perhaps resulting from Boolean operators). Being a polyglot at heart, I have enjoyed learning such an unusual language as APL.

Bibliography

1. Iverson, Kenneth E., *A Programming Language*, John Wiley & Sons, Inc., New York, 1962.
2. Peelle, Howard A., *APL: An Introduction*, Hayden Book Company, Inc., Rochelle Park, New Jersey, 1978.
3. Gilman, Leonard and Rose, Allen J., *APL: An Interactive Approach*, John Wiley & Sons, Inc., New York, 1976.
4. Wiedmann, Clark, *Handbook of APL Programming*, Petrocelli Books, New York, 1974.
5. Katzan, Jr., Harry, *APL Programming and Computer Techniques*, Van Nostrand Reinhold Company, New York, 1970.
6. Grey, Louis D., *A Course in APL with Applications*, Addison-Wesley Publishing Company, Inc., Philippines, 1976.

You may contact Mr. Peterson at 8628 Edgehill Ct., El Cerrito, CA 94530.

MICRO™

What's Where in the Apple

A Complete Guide to the Apple Computer

This **REVISED EDITION** of the famous Apple Atlas provides Apple computerists with a framework for understanding both the overall organization and structure of the Apple system and programming techniques that exploit that knowledge.

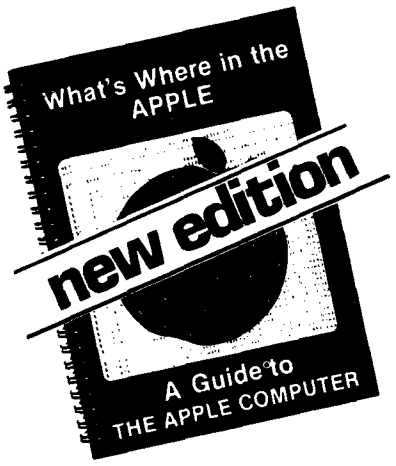
What's Where in the Apple contains the most complete memory map ever published as well as detailed information needed for actual programming.

All for only $24.95
(plus $5.00 s/h)

For owners of the original edition, MICRO is offering a companion book, ***THE GUIDE to What's Where in the Apple,*** for only **$9.95** (plus $2.00 s/h)

THE GUIDE contains all new material that explains and demonstrates how to use the atlas and gazetteer published in the original volume of *What's Where in the Apple?*

MICRO makes it easy to order: Send check (payable to MICRO) to:

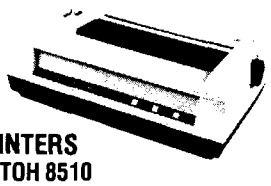


VISA and MasterCard accepted
MA residents add 5%

MICRO INK P.O. Box 6502 Chelmsford, MA 01824
Call our **toll-free** number: **1-800-354-8112** (In PA, 1-800-662-2444)

FOR YOUR APPLE II

Industry standard products at super saver discount prices



PARALLEL PRINTERS
NEC 8023 or C-ITOH 8510

(Virtually identical) Specifications: • 100 CPS dot matrix printer • 80 column print–136 characters per line • Tractor/friction feed • 7 different print fonts included • 2K printer buffer • Proportional spacing • Bit image graphics and graphic symbols.

NEC 8023 or C-ITOH **$495**
NEC 8023 or C-ITOH 8510 with Parallel Interface and Cable **$550**
EPSON 100 with Parallel Interface and Cable **$749**

Z-80 CARD FOR YOUR APPLE
MICROSOFT SOFTCARD

With CP/M* and MBASIC.

(List: $399) **$289**

NEW! *Best Buy!!!*
ADVANCED LOGIC SYSTEM Z-CARD With C-PM*

Has everything the Softcard has except MBASIC. Works with Microsoft's disks too.

(List $269) ..... **Special at $195**

NEW!
ALS SYNERGIZER

CP/M* operating package with an 80 column video board, CP/M* interface, and 16K memory expansion for Apple II. Permits use of the full range of CP/M* software on Apple II. Includes SuperCALC.

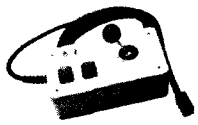
(List: $749) **$549**

NEW!
U-Z-80 PROCESSOR BOARD (From Europe)

Software compatible with Softcard and ALS Software **$149**

MICROSOFT + PREMIUM SYSTEM

Includes Videx Videoterm, Softswitch, Microsoft and Softcard, Microsoft and Z-80 Card, and Osborn CP/M* Manual **$595**



JOYSTICK

Takes the place of two Apple Paddle Controllers.

From BMP Enterprises. Heavy duty industrial construction and cable. Non-self centering. With polarity switches for consistent motion control.

(List: $59) **$39**

MONITORS FOR YOUR APPLE

AMDEK 300G (18MHZ Anti-Glare Screen) **$179**
NEC 12" HIRES GREEN **$179**

SUPER SPECIAL!
SPECIAL 12" GREEN MONITOR **$99**

SPECIAL AND NEW

5 MEGABYTE HARD DISK

For Apple II. Supplied with controller. Use with CP/M, Apple DOS, & Apple Pascal **$1995**

5¼" DISK DRIVE

Use with standard Apple II disk controller. ..**$295**

5¼" FLOPPY DISKS

With hub rings. Box of 10.

With other purchase **$19.95**
Without purchase **$23.00**

16K MEMORY EXPANSION MODULE

The preferred 16K RAM Expansion Module from PROMETHEUS. Fully compatible with CP/M* and Apple Pascal*. With full 1-year parts and labor warranty. (List: $169) **$75**

WORD PROCESSING SPECIAL
WITH WORDSTAR AND SUPERCALC!

Do professional word processing on your APPLE. All necessary hardware and software included. Complete 80 column video display, enhanced character set, 16K memory board, Z-Card with CP/M* software, Wordstar and word processing software and SuperCALC.

(List: $1,128) ....**Special at $695**

NEW! *from Prometheus!*
ExpandaRAM

The only 128K RAM card that lets you start with 16K, 32K, or 64K of memory now and expand to the full 128K later. Fully compatible with Apple Pascal, CP/M*, and Visacalc. No Apple modification required. Memory management system included with all ExpandaRAMs. Disk emulators included with 64K and 128K versions.

MEM-32 Two rows of 16K RAMS make a 32K RAM Card **$209**
MEM-64 One row of 64K RAM. With DOS 3.3 disk emulator **$299**
MEM-128 Two rows of 64K RAMS installed make a 128K Card. With DOS 3.3 disk emulator **$399**
MEM-RKT 64K RAM Add-On-Kits– 64K Dynamic RAMS. Each **$125**
VISICALC Expansion Program for MEM-128 ..... **$75**
MEM-PSL Pascal disk emulator for MEM-128 ..... **$45**

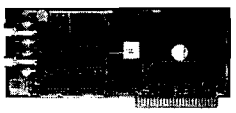
MODEMS FOR YOUR APPLE II

HAYES Smartmodem **$229**
MICROMODEM II **$279**

VERSAcard FROM PROMETHEUS

Four cards on one! With true simultaneous operation. Includes: (1) Serial Input/Output Interface, (2) Parallel Output Interface, (3) Precision Clock/Calendar, and (4) BSR Control. All on one card. Fully compatible with CP/M* and Apple Pascal*.

(List: $249) **$169**



80 COLUMN VIDEO DISPLAYS FOR APPLE II
SMARTERM

(Not to be confused with SUPRTERM)

Software switching from 80 to 40 and 40 to 80 characters. 9 new characters not found on the Apple keyboard. Fully compatible with CP/M* and Apple PASCAL*. With lowest power consumption of only 2.5 watts.

(List: $345) **$225**

SMARTERM EXPANDED CHARACTER SET

7" x 11" matrix with true decenders. Add to above **$40**

Best Buy! **Combination SMARTERM and EXPANDED CHARACTER SET**

Special at **$260**
VIDEX, VIDEOTERM **$249**
VIOEX ENHANCER II **$119**

NEW!
CENTRONICS COMPATIBLE PARALLEL INTERFACE

From PROMETHEUS. For use with Epson, NEC, C-ITOH, and other printers. Fully compatible with CP/M® and Apple Pascal®.

PRT-1, Only **$69**

GRAPHITTI CARD

Prints HIRES page 1 or 2 from onboard firmware. Features: True 1:1 aspect ratio, prints emphasized mode, reverse mode, rotates 90 degrees . . . plus more. Compare all this with the Grappler. We think you'll agree that this is the best graphics card on the market. Specify for use with EPSON, NEC-8023, C-ITOH Prowriter, or Okidata.

(List: $125) **$89**

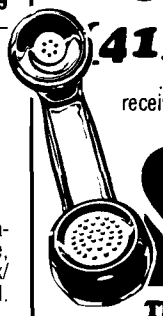
SOFTWARE

WORDSTAR ..... **Special at $195**
SPELLSTAR **$125**
SUPERCALC **$175**
D BASE II **$525**
VISICALC **$149**
DB MASTER **$189**

All equipment shipped factory fresh. Manufacturers' warranties included. Please add $3.00 per product for shipping and handling. California: add 6% tax; BART Counties: 6½%.

All items are normally in stock

Phone for Quick Shipment!



(415) 490-3420

. . . And we'll be here to help after you receive your order. Feel free to call the SGC Technical Staff for assistance.

SGC

The mail order specialists

342 Quartz Circle, Livermore, CA 94550

EDIT: An ATARI FORTH Screen-Oriented Editor

by Mike Dougherty

EDIT uses the Atari 800 display as a text window into a FORTH disk screen, and allows full use of the Atari special function keys to prepare FORTH applications.

EDIT
requires:
Atari 800 with 24K
Atari 810 disk drive
APX fig-FORTH

The Atari Program Exchange version of FORTH, "APX fig-FORTH," comes with the sources for two line-oriented editors on disk. Although line editing is greatly enhanced by using the Atari screen edit keys, a general video screen editor makes software development easier. EDIT implements a video screen editor for the Atari 800. This 2.6K-byte application can be modified to run on most memory-mapped video FORTH systems. With modification of the low-level terminal words, EDIT should be adaptable to FORTH systems containing a serial terminal with an addressable cursor.

EDIT Design

The two objectives of EDIT are to provide a useful video screen editor and to maintain full compatibility with APX fig-FORTH. Specifically, the following points are considered:

1. Retaining FORTH's 16-line by 64-character text screen.
2. Making FORTH LISTs or TRAIDs of edited screens visually acceptable.
3. Allowing screens to be compiled by a FORTH LOAD.
4. Making the video reflect the current state of the text.
5. Causing the video screen editor to execute with the Atari's default 38-character line length.

***Listing 1:* EDIT for Atari APX fig-FORTH**

```
SCR # 60
  0 (          E D I T                 )
  1 (                                  )
  2 ( APX fig-Forth  Screen Editor     )
  3 (                                  )
  4 (       by  Mike Dougherty         )
  5 (                                  )
  6 ( This editor allows the user      )
  7 ( to edit a Forth screen using     )
  8 ( the Atari display as a text      )
  9 ( window into the Forth screen     )
 10 (                                  )
 11 ( To load EDIT:        60 LOAD     )
 12 (                                  )
 13 ( To EDIT screen #n:    n EDIT     )
 14
 15 -->


SCR # 61
  0 ( CONSTANTS / VARIABLES FOR EDITOR )
  1
  2  0 VARIABLE CLINE                  ( Current line number, 0-15    )
  3  0 VARIABLE CCHAR                  ( Current char number, 0-63    )
  4  3 VARIABLE LINEOFF                ( Offset for 1ST text line     )
  5  3 VARIABLE CHAROFF                ( Offset for left margin       )
  6 18 CONSTANT TOP-BOT                ( Top & bottom window char     )
  7  2 CONSTANT REDGE                  ( Right edge window char       )
  8 22 CONSTANT LEDGE                  ( Left edge window char        )
  9  0 VARIABLE SIDE                   ( Current side of screen       )
 10 17 VARIABLE EXTRAOFF               ( Extra line screen position   )
 11
 12  0 VARIABLE STOP                   ( Editor exit flag             )
 13  0 VARIABLE SAV-BUF 64 ALLOT       ( Buffer for deleted line      )
 14
 15 -->


SCR # 62
  0 ( READ SCREEN SCR INTO MEMORY )
  1
  2 : EREAD                            ( --- )
  3
  4     16 0 DO                        ( For each line of the screen )
  5       I SCR @ (LINE)               ( Read & get the ADDR, LENGTH )
  6       DROP UPDATE DROP             ( Mark the block of memory )
  7     LOOP                           ( next screen line )
  8
  9     0 CLINE !                      ( Initialize to 1ST LINE )
 10     0 CCHAR !                      ( ...         to 1ST CHAR )
 11     0 SIDE  ! ;                    ( ...         to LEFT SIDE )
 12
 13
 14
 15 -->

SCR # 63
  0 ( PRIMATIVE SCREEN MANIPULATIONS )
  1
  2 : ECLEAR                           ( --- )
  3     XGR                            ( Clear video via GRAPHICS 0 )
  4     1 752 C! ;                     ( Inhibit ATARI system cursor )
  5
  6 : POINT-CURSOR                     ( row col --- )
  7     85 !                           ( Save col in system shadow )
  8     84 C! ;                        ( Save row too -- only 1 byte )
  9
 10 : CURSOR@                          ( --- row col )
 11     CLINE @ LINEOFF @ +            ( Compute video position )
 12     CCHAR @                        ( Base value for col )
 13     SIDE @ IF 32 - ENDIF           ( Adjust if RIGHT side )
 14     CHAROFF @ + ;                  ( Add left margin )
 15 -->
```

6. Performing logically equivalent functions on the screen text with all special edit keys of the Atari 800.

To use the Atari display, and maintain compatibility, EDIT uses a text window of 16 lines by 32 characters. A FORTH screen is thus divided in half — the left side (SIDE = 0) and the right side (SIDE = 1). The current side of the FORTH screen is displayed on the video screen with a solid line around the text area. The current text position is indicated by inverse video, as in the normal Atari display. In addition to the text window, the other half of the current line is displayed at the bottom of the video screen.

In my applications, the left side of a FORTH screen is used for actual FORTH code and the right side is reserved for comments. Thus, when viewing a FORTH screen, all the code on the left side (SIDE = 0) may be examined at once, while only one comment line is displayed at a time as the cursor is moved from line to line. So far, 32 characters have been enough to code a logical FORTH step, and 32 characters are usually adequate to comment that step. In addition, this interpretation of a FORTH screen encourages a vertical style of FORTH definitions, with a comment for each step. Considering the low cost of diskettes, I prefer to spend a small amount of money for the comments of a vertical definition, rather than to spend a large amount of time deciphering terse, horizontal definitions, containing few step-by-step comments.

A final pragmatic reason for selecting the 32-character text window lies in the fig-FORTH treatment of disk blocks when read into memory. Consecutive fig-FORTH disk blocks are not necessarily stored in memory consecutively. Further, a disk block in memory must also contain disk-related information. Fortunately, after a disk block is read into memory, the starting address of any 64-character line may be retrieved by the FORTH word (LINE). Since the characters of each line are stored consecutively in memory, a 32-character text window is easily manipulated. A text window not fitting evenly into 64 characters would have to be handled by overlapping from block to block.

Implementation

EDIT is implemented as a turn-key application. That is, once EDIT is in-

Listing 1 *(continued)*

```
SCR # 64
  0 ( PRIMITIVE SCREEN MANIPULATIONS)
  1
  2 : CURSOR                          ( --- )
  3     CURSOR@                       ( Get video row and col pos )
  4     POINT-CURSOR ;                ( Point ATARI text cursor )
  5
  6 : SCREEN-CURSOR                   ( --- )
  7     CURSOR@                       ( Get video row and col pos )
  8     SWAP 40 * +                   ( Form GR.0 mem addr offset )
  9     106 C@ 256 *                  ( Get top of memory addr )
 10     960 -                         ( Backup to 1st of display )
 11     + DUP                         ( Form mem addr of cursor )
 12     C@ 128 XOR                    ( Get screen display, invert )
 13     SWAP C! ;                     ( Put inverse video back )
 14
 15 -->


SCR # 65
  0 ( PRIMITIVES FOR DISPLAY )
  1
  2 : HLINE                           ( --- )
  3     34 0 DO                       ( For horizontal window span )
  4       TOP-BOT EMIT                ( Put a bar in each char )
  5     LOOP ;
  6
  7 : LADDR                           ( --- addr )
  8     CLINE @ SCR @ (LINE) DROP     ( Compute mem addr of line )
  9     SIDE @ IF 32 + ENDIF ;        ( Adjust for left side )
 10
 11 : EADDR                           ( --- addr )
 12     CLINE @ SCR @ (LINE) DROP     ( Compute mem addr of line )
 13     SIDE @ 0= IF 32 + ENDIF ;     ( Adjust for right side )
 14
 15 -->


SCR # 66
  0 ( PRIMITIVES FOR DISPLAY )
  1
  2 : ELINE                           ( --- )
  3     LEDGE EMIT                    ( Output left window edge )
  4     LADDR 32 TYPE                 ( Type actual text line )
  5     REDGE EMIT ;                  ( Output right window edge )
  6
  7 : XLINE                           ( --- )
  8     LEDGE EMIT                    ( Output left window edge )
  9     EADDR 32 TYPE                 ( Output text line overflow )
 10     REDGE EMIT ;                  ( output right window edge )
 11
 12
 13
 14
 15 -->


SCR # 67
  0 ( PRIMITIVES FOR DISPLAY )
  1
  2 : WRITE-LINE                      ( --- )
  3     CLINE @ LINEOFF @ +           ( Get video line number )
  4     CHAROFF @ 1 -                 ( Point to left border pos )
  5     POINT-CURSOR                  ( Point ATARI text cursor )
  6     ELINE ;                       ( Output edit line here )
  7
  8 : WRITE-EXTRA                     ( --- )
  9     EXTRAOFF @ LINEOFF @ +        ( Get video line number )
 10     CHAROFF @ 1 -                 ( Point to left border pos )
 11     POINT-CURSOR                  ( Point ATARI text cursor )
 12     XLINE ;                       ( Output extra line here )
 13
 14
 15 -->


SCR # 68
  0 ( PRIMITIVES FOR DISPLAY )
  1
  2 : TOP                             ( --- )
  3     LINEOFF @ 1 -                 ( Before 1st line video )
  4     CHAROFF @ 1 -                 ( Position of left window )
  5     POINT-CURSOR                  ( Point ATARI text window )
  6     HLINE ;                       ( Output a horizontal line )
  7
  8 : BOT                             ( --- )
  9     LINEOFF @ 16 +                ( After last line video )
 10     CHAROFF @ 1 -                 ( Position of left window )
 11     POINT-CURSOR                  ( Point ATARI text window )
 12     HLINE ;                       ( Output a horizontal line )
 13
 14
 15 -->
```

Listing 1 *(continued)*

```
SCR # 69
  Ø ( PRIMITIVES FOR DISPLAY )
  1
  2 : ETITLE                          ( --- )
  3     Ø CHAROFF @ POINT-CURSOR      ( Point to 1st video line )
  4     ." Screen: " SCR ?            ( Type the screen number )
  5     1 CHAROFF @ POINT-CURSOR      ( Point to 2nd video line )
  6     ." Side:   " SIDE ? ;         ( Type the screen number )
  7
  8
  9
 10
 11
 12
 13
 14
 15 -->


SCR # 70
  Ø ( DISPLAY CURRENT SCREEN )
  1
  2 : DISPLAY                         ( --- )
  3     ECLEAR ETITLE                 ( Clear video, put title )
  4     TOP                           ( Output top of window )
  5     CLINE @                       ( Save current line on stack )
  6     16 Ø DO                       ( For each line in text )
  7       I CLINE ! WRITE-LINE        ( Set CLINE and output it )
  8     LOOP                          ( Next Forth text line )
  9     CLINE !                       ( Restore current line )
 10     SIDE @ IF 32 ELSE Ø ENDIF     ( Get start depending on side )
 11     CCHAR !                       ( Store as current col pos )
 12     BOT                           ( Output bottom of window )
 13     WRITE-EXTRA                   ( Output current line ovrflow )
 14     CURSOR SCREEN-CURSOR ;        ( Position cursor, show it )
 15 -->


SCR # 71
  Ø ( CASE STATEMENT BY DR. C. E. EAKER, FORTH DIMENSIONS [V2,#3]  )
  1
  2 : DOCASE  ?COMP CSP @ !CSP 4 ; IMMEDIATE
  3
  4 : CASE 4 ?PAIRS COMPILE OVER COMPILE = COMPILE ØBRANCH
  5   HERE Ø , COMPILE DROP 5 ; IMMEDIATE
  6
  7 : ENDCASE 5 ?PAIRS COMPILE BRANCH HERE Ø ,
  8   SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
  9
 10 : ENDCASES 4 ?PAIRS COMPILE DROP
 11   BEGIN SP@ CSP @ = Ø= WHILE
 12     2 [COMPILE] ENDIF REPEAT
 13   CSP ! ; IMMEDIATE
 14
 15 -->


SCR # 72
  Ø ( CURSOR MOVEMENT PRIMITIVES )
  1
  2 : MOVE-RIGHT                      ( --- )
  3     CCHAR @ 1 +                   ( Get/increment char pos )
  4     SIDE @ IF                     ( Handle wrap around on side )
  5       DUP 64 = IF DROP 32 ENDIF   ( Over 63 goes to 32 on right )
  6     ELSE
  7       DUP 32 = IF DROP Ø ENDIF    ( Over 31 goes to Ø on left )
  8     ENDIF
  9     CCHAR ! ;                     ( Store new current char )
 10
 11 : RIGHT                           ( --- )
 12     MOVE-RIGHT SCREEN-CURSOR ;    ( Move and set cursor )
 13 : RIGHT-CURSOR                    ( --- )
 14     SCREEN-CURSOR RIGHT ;         ( Restore cursor, move )
 15 -->


SCR # 73
  Ø ( CURSOR MOVEMENT PRIMITIVES )
  1
  2 : MOVE-LEFT                       ( --- )
  3     CCHAR @ 1 -                   ( Get/decrement current pos )
  4     SIDE @ IF                     ( Wrap around on side )
  5       DUP 31 = IF DROP 63 ENDIF   ( Go to 63 on right wrap )
  6     ELSE
  7       DUP -1 = IF DROP 31 ENDIF   ( Go to 31 on left wrap )
  8     ENDIF
  9     CCHAR ! ;                     ( Store new char pos )
 10
 11 : LEFT                            ( --- )
 12     MOVE-LEFT SCREEN-CURSOR ;     ( Move and set new cursor )
 13 : LEFT-CURSOR                     ( --- )
 14     SCREEN-CURSOR LEFT ;          ( Restore cursor, move )
 15 -->
```

voked, it interprets all user keystrokes until the end of the edit session. In general, the words called by EDIT should not be executed from the keyboard. Since the EDIT words manipulate the screen, results of direct execution can be confusing. EDIT is invoked by pushing the screen number on the data stack and calling EDIT. For example, to edit screen number 60, type:

```
60 EDIT
```

Listing 1 is the EDIT application occupying screens 60 through 86 on my EDIT source disk. The original EDIT text was entered with the APX fig-FORTH editor [27 LOAD] and rewritten with the debugged version of EDIT.

The basic structure of EDIT is straightforward. As defined on screens 85 and 86, EDIT reads a screen into memory, displays the left half [SIDE = 0] on the video display, and sets a stop flag to zero [no stop]. The main loop is executed until the stop flag is set to a non-zero value by one of the FORTH words EXIT, EABORT, ENEXT, or ELAST. EDIT inputs a terminal key each pass through the main

COMPU SENSE

CARDBOARD 3
An Economy Expansion Interface
(Motherboard)
For the VIC-20® Personal Computer

The "CARDBOARD/3" is an expansion interface designed to allow the user to access more than one of the plug-in-type memory or utility cartridges now available. It will accept up to 3 RAM or ROM cartridges at once. For example:

- 16k RAM + 16k RAM + 3k RAM
- 16k RAM + 8k RAM + Super Expander
- 16k RAM + 8k RAM + Vic-Mon
- 16k RAM + 3k RAM + Programmer's Aid
- High quality T.R.W. gold plated connectors
- This board is fused
- 90 day free replacement warranty covering everything except the fuse

$39.95

CARDBOARD 6
An Expansion Interface for VIC-20®

- *Allows memory expansion up to 40K*
- Accepts up to six games
- Includes a system reset button
- *All slots are switch selectable*
- Daisy chain several units for even more versatility

$87.95

TO ORDER:
P. O. BOX 18765
WICHITA, KS 67218
(316) 684-4660

Personal checks accepted
(Allow 3 weeks) or
C.O.D. *(Add $2)*
Handling charge $2.00

VIC-20™ is a registered trademark of Commodore

loop. If the key is a special case, EDIT executes the corresponding special function. Otherwise the key is added to the FORTH screen and the video screen. CASE structure allows EDIT to be modified or expanded easily, yet executes quickly.

A two-key escape sequence is used to add special functions, which do not have Atari keys, to EDIT. The first escape character starts the execution of the FORTH word ESC. ESC, like EDIT, uses a CASE statement to allow the next key input to select a special function. Different classes of special functions can be added easily to EDIT with this technique.

This is one of my first APX fig-FORTH applications and the expert FORTH coder may notice how little the stack is used as word inputs. Old programming habits die hard! Even so, there are only four important variables used in EDIT. CLINE and CCHAR maintain the current line and current character position in memory and on the text window display. SIDE keeps track of whether the left side (SIDE = 0) or the right side (SIDE = 1) of the FORTH screen is displayed in the text

SYSTEMS INTEGRATOR

INTRODUCING:

ZYTREX
ZT14411
CMOS BAUD RATE
GENERATOR

REPLACES MOTOROLA MC14411

- PIN/FUNCTION COMPATIBLE
- IMPROVED FREQ OUTPUT DRIVE (4 LSTTL LOADS)
- FULLY STATIC OPERATION
- TTL-COMPATIBLE INPUTS
- WIDE OPERATING VOLTAGE

FREE EVALUATION SAMPLES FOR VOLUME USERS

$6.20 EACH AT 1000 PCS.

ZYTREX CORPORATION
224 NORTH WOLFE ROAD
SUNNYVALE, CA 94086
(408) 733-3973

Listing 1 *(continued)*

```
SCR # 74
  0 ( CURSOR MOVEMENT PRIMITIVES )
  1
  2 : MOVE-UP                          ( --- )
  3     CLINE @ 1 - 15 AND             ( Adjust current line, 0-15 )
  4     CLINE !                        ( Save - wraparound by AND )
  5     WRITE-EXTRA ;                  ( Add extra line at bottom )
  6
  7 : EUP                              ( --- )
  8     MOVE-UP                        ( Move up the display )
  9     SCREEN-CURSOR ;                ( Inverse video cursor pos )
 10
 11 : UP-CURSOR                        ( --- )
 12     SCREEN-CURSOR                  ( Restore current cursor )
 13     EUP ;                          ( move up, new cursor )
 14
 15 -->

SCR # 75
  0 ( CURSOR MOVEMENT PRIMITIVES )
  1
  2 : MOVE-DOWN                        ( --- )
  3     CLINE @ 1 + 15 AND             ( Move down a line, 0-15 )
  4     CLINE !                        ( Save new, wraparound by AND )
  5     WRITE-EXTRA ;                  ( Add extra text line )
  6
  7 : DOWN                             ( --- )
  8     MOVE-DOWN                      ( Move down a line )
  9     SCREEN-CURSOR ;                ( Set new cursor video pos )
 10
 11 : DOWN-CURSOR                      ( --- )
 12     SCREEN-CURSOR                  ( Restore video cursor pos )
 13     DOWN ;                         ( Move down, set new cursor )
 14
 15 -->


SCR # 76
  0 ( EDITOR PRIMITIVES )
  1
  2 : RETURN                           ( --- )
  3     SCREEN-CURSOR                  ( Restore cursor video )
  4     SIDE @ IF 32 ELSE 0 ENDIF      ( Get beginning of line )
  5     CCHAR !                        ( Set as the current char )
  6     DOWN ;                         ( Move down )
  7
  8 : EXIT                             ( --- )
  9     FLUSH                          ( Force disk output of update )
 10     1 STOP ! ;                     ( Set editor stop flag )
 11
 12 : EABORT                           ( --- )
 13     EMPTY-BUFFERS                  ( Scratch updated buffers )
 14     1 STOP ! ;                     ( Set editor stop flag )
 15 -->


SCR # 77
  0 ( EDITOR PRIMITIVES )
  1
  2 : ENEXT                            ( --- )
  3     FLUSH                          ( Write out updated buffers )
  4     SCR @ 1 + SCR !                ( Next screen, no error check )
  5     EREAD                          ( Read the FORTH screen )
  6     DISPLAY ;                      ( Display, and edit )
  7
  8 : ELAST                            ( --- )
  9     FLUSH                          ( Write out updated buffers )
 10     SCR @ 1 - SCR !                ( Last screen, no error check )
 11     EREAD                          ( Read the FORTH screen )
 12     DISPLAY ;                      ( Display, and edit )
 13
 14
 15 -->


SCR # 78
  0 ( EDITOR PRIMITIVES )
  1
  2 : ADDR                             ( --- )
  3     CLINE @                        ( Get the current line number )
  4     SCR @ (LINE) DROP              ( Get the memory address )
  5     CCHAR @ + ;                    ( Add the current char pos )
  6
  7 : ADDKEY                           ( --- )
  8     DUP ADDR C!                    ( Store a copy in memory text )
  9     CURSOR EMIT                    ( Output char to video screen )
 10     RIGHT ;                        ( Move right for next )
 11
 12 : CHANGE-SIDE                      ( --- )
 13     SIDE @ 1 XOR SIDE !            ( Flip LSB: 0,1 legal values )
 14     DISPLAY ;                      ( Display, set CCHAR to start )
 15 -->
```

Listing 1 *(continued)*

```
SCR # 79
  Ø ( EDITOR PRIMITIVES )
  1
  2 : INSERT-CHAR                    ( end-of-line --- )
  3     ADDR SWAP CCHAR @ -          ( Current addr, chars to move )
  4     SWAP OVER + SWAP             ( End of line, # to move )
  5     -DUP IF                      ( If any chars to move right )
  6       Ø DO                       ( For each char to move over )
  7         DUP 1 - C@ OVER C!       ( Store previous toward end )
  8         1 -                      ( Next one closer to blank )
  9       LOOP                       ( Until space opened in line )
 1Ø     ENDIF
 11     BL SWAP C! ;                 ( Fill for space )
 12 : INSERT                        ( --- )
 13     SIDE @ IF 63 ELSE 31 ENDIF   ( Set end of line limit )
 14     INSERT-CHAR WRITE-LINE       ( Insert blank, output line )
 15     CURSOR SCREEN-CURSOR ; -->   ( Set the cursor )


SCR # 8Ø
  Ø ( EDITOR PRIMITIVES )
  1
  2 : DELETE-CHAR                    ( end-of-line --- )
  3     ADDR SWAP CCHAR @ -          ( Get addr, # chars to move )
  4     -DUP IF                      ( If any chars to move left )
  5       Ø DO                       ( For each moving character )
  6         DUP 1 + C@ OVER C!       ( Move right to left )
  7         1 +                      ( Next address in line )
  8       LOOP
  9     ENDIF
 1Ø     BL SWAP C! ;                 ( Insert blank at end of line )
 11
 12 : DELETE                        ( --- )
 13     SIDE @ IF 63 ELSE 31 ENDIF   ( Determine end char of line )
 14     DELETE-CHAR WRITE-LINE       ( Delete current char, outut )
 15     CURSOR SCREEN-CURSOR ; -->   ( Reset cursor )

SCR # 81
  Ø ( EDITOR PRIMITIVES )
  1
  2 : L-A                            ( --- addr )
  3    SCR @ (LINE) DROP ;           ( Get the address of start )
  4
  5 : DELETE-L                       ( --- )
  6    CLINE @ L-A SAV-BUF 64 CMOVE  ( Save current in SAV-BUF )
  7    15 CLINE @ - IF               ( If not the last line )
  8      15 CLINE @ DO               ( For each line above current )
  9        I 1+ L-A I L-A 64 CMOVE   ( Move line one toward line#Ø )
 1Ø      LOOP
 11    ENDIF
 12    15 L-A 64 BLANKS ;            ( Blank line # 15 )
 13
 14 : DELETE-LINE                   ( --- )
 15    DELETE-L DISPLAY ; -->        ( Delete and display )


SCR # 82
  Ø ( EDITOR PRIMITIVES )
  1
  2 : INSERT-L                       ( --- )
  3    15 CLINE @ - IF               ( If not the last line )
  4      CLINE @ 15 DO               ( Start at the end of screen )
  5        I 1 - L-A I L-A 64 CMOVE  ( Shuffle lines to end )
  6      -1 +LOOP                    ( Move toward current )
  7    ENDIF
  8    CLINE @ L-A 64 BLANKS ;       ( Blank current line )
  9 : INSERT-LINE                    ( --- )
 1Ø    INSERT-L DISPLAY ;            ( Insert blank line, display )
 11
 12 : PUT-LINE                      ( --- )
 13    INSERT-L                      ( Insert a blank line )
 14    SAV-BUF CLINE @ L-A 64 CMOVE  ( Restore deleted line )
 15    DISPLAY ;  -->                ( Display )


SCR # 83
  Ø ( EDITOR PRIMITIVES )
  1
  2 : TAB                            ( --- )
  3     2 Ø DO                       ( Tab spaces only 2 for me )
  4       RIGHT-CURSOR               ( Simulate moving cursor )
  5     LOOP ;
  6
  7 : DEL                            ( --- )
  8     LEFT-CURSOR BL DUP ADDR C!   ( Move left and blank mem )
  9     CURSOR EMIT SCREEN-CURSOR ;  ( Echo to screen )
 1Ø
 11 : CLEAR-SCR                      ( --- )
 12     16 Ø DO                      ( For all 15 lines in screen )
 13       I L-A 64 BLANKS            ( Blank memory text )
 14     LOOP
 15     DISPLAY ;   -->              ( Display results )
```

window. Finally, STOP is the exit flag used in the main loop (0 = continue, 1 = stop).

EDIT implements the following special function keys of the Atari 800:

Command	Function
[delete]	Delete character before cursor
CTRL/[delete]	Delete current character, shrink line in text window
SHFT/[delete]	Delete current line, move rest up one line
CTRL/[insert]	Insert space at cursor, expand line in text window
SHFT/[insert]	Insert new line at cursor, line 15 is lost from the display but saved in a buffer for the "ESC P" command
CTRL/[clear]	Clear the text window
SHFT/[clear]	Clear the text window
CTRL/[right]	Move cursor right within text window
CTRL/[left]	Move cursor left within text window
CTRL/[up]	Move cursor up within text window
CTRL/[down]	Move cursor down within text window
RETURN	Advance to the first character of the next line
TAB	Move two characters right

Only the text window is affected by these special function keys.

The following functions are implemented as two-key escape sequences:

Command	Function
ESC E	Exit the editor, writing the edited screen to disk
ESC A	Abort the edit session, no change to the disk screen
ESC N	Save the current screen, edit the next screen
EXC L	Save the current screen, edit the last screen
ESC S	Switch FORTH screen sides in the text window
ESC P	Put down (insert) the last line deleted

The escape sequence method was chosen in order to leave the normal and

Lyco Computer Marketing & Consultants

TO ORDER CALL US

TOLL FREE 800-233-8760

In PA 1-717-398-4079

FEBRUARY ATARI SPECIALS

810 Disk Drive ... $429.00
400 32K RAM ... $CALL$

NEW ATARI COMPUTER ... $ CALL $

800 48K ... $499.00

ATARI®
A Warner Communications Company

ATARI HARDWARE

410 CASSETTE RECORDER	$ 75.00
825 PRINTER	$585.00
830 PHONE MODEM	$149.00
850 INTERFACE	$164.00

PACKAGES

CX481 ENTERTAINER	$ 69.00
CX482 EDUCATOR	$125.00
CX483 PROGRAMMER	$ 49.00
CX494 COMMUNICATOR	$325.00

SOFTWARE

CXL4012 MISSILE COMMAND	$28.75
CXL4013 ASTEROID	$28.75
CXL4020 CENTIPEDE	$32.75
CXL4022 PACMAN	$32.75
CXL4011 STAR RAIDER	$34.75
CXL4004 BASKETBALL	$26.75
CXL4006 SUPER BREAKOUT	$28.75
CXL4008 SPACE INVADER	$28.75
CX8130 CAVERNS OF MARS	$31.75
CX4108 HANGMAN	$12.75
CX4102 KINGDOM	$12.75
CX4112 STATES & CAPITALS	$12.75
CX4114 EUROPEAN COUNTRIES	$12.75
CX4109 GRAPHIT	$16.75
CX4121 ENERGY CZAR	$12.75
CX4123 SCRAM	$19.75
CX4101 PROGRAMMING I	$19.75
CX4106 PROGRAMMING II	$22.75
CX4117 PROGRAMMING III	$22.75
CXL4015 TELELINK	$21.75
CX4119 FRENCH	$39.75
CX4118 GERMAN	$39.75
CX4120 SPANISH	$39.75
CXL4007 MUSIC COMPOSER	$33.75
CXL4002 ATARI BASIC	$45.75
CX8126 MICROSOFT BASIC	$65.75
CXL4003 ASSEMBLER EDITOR	$45.75
CX8126 MACRO ASSEMBLER	$69.75
CXL4018 PILOT HOME	$65.75
CX405 PILOT EDUCATOR	$99.75
CX415 HOME FILING MANAGER	$41.75
CX414 BOOKKEEPER	$119.75

NEW RELEASES

CHOP LIFTER	$27.75
APPLE PANIC	$23.75
PREPPIE	$19.95

THIRD PARTY SOFTWARE

EASTERN FRONT 1941	$25.50
OUTLAW/HOWITZER	$15.50
WIZARD of WAR	$31.00
GORF	$31.00
FROGGER	$26.00
CHOP LIFTER	$27.75
APPLE PANIC	$23.75
PREPPIE	$19.95
STAR WARRIOR	$28.00
CRUSH,CRUMBLE,&CHOMP	$23.00
SHOOTING GALLERY	$19.95
VIDEO MATH FLASH	$12.00
MY FIRST ALPHABET	$25.50
BAHA BUGGIES	$24.95
TEMPLE of ASPHAI	$27.95
UPPER REACHES of ASPHAI	$15.00
TRACK ATTACK	$23.00
STAR BLAZER	$25.00
LABYRINTH	$23.00
SEA FOX	$23.00
POOL 1.5	$26.95
SPEEDWAY BLAST (ROM)	$29.95
JAWBREAKER	$22.95
THRESHOLD	$29.95
MOONBASE IO	$23.95
PROTECTOR	$24.95
NAUTILUS	$24.95
SLIME	$24.95
SUBMARINE COMMANDER (ROM)	$36.95
JUMBO JET PILOT (ROM)	$36.95
SOCCER (ROM)	$36.95
KICKBACK (football ROM)	$36.95

PRINTERS

Okidata 82A	$479.00
Okidata 83A	$719.00
Okidata 84	$1089.00
Citoh	CALL
Prowriter I	$499.00
Prowriter II	CALL
SMITH CORONA TP-1	$625.00
NEC	CALL

(Interfacing Available)

BUSINESS SOFTWARE

ATARI WORD PROCESSING	$109.00
LETTER PERFECT (ROM)	$149.00
LETTER PERFECT (disc)	$129.00
TEXT WIZZARD	$ 89.00
DATA PERFECT	$ 75.00
VISICALC	$169.00
DATASAM/65	$125.00

JOYSTICKS

ATARI CX-40	$18.00
LESTICK	$34.00
WICO COMMAND CONTROL	$23.75
WICO RED BALL	$26.75
WICO TRACK BALL	$54.75
STICK STAND	$ 6.75

COMPUTER COVERS

800	$6.99
400	$6.99
410	$6.99
810	$6.99

PERCOM

SINGLE DRIVE (SD)	$399.00
SINGLE DRIVE (DD)	$549.00
DUAL DRIVE (DD)	$869.00
DUAL HEAD (DD)	$669.00

POLICY

In-Stock items shipped within 24 hours of order. Personal checks require four weeks clearance before shipping. No deposit for COD orders. PA residents add sales tax. All products subject to availability and price change. Advertised prices show 4% discount offered for cash. Add 4% for Mastercard and Visa.

TO ORDER
CALL TOLL FREE
800-233-8760
In PA 1-717-398-4079
or send order to
Lyco Computer
P.O. Box 5088
Jersey Shore, PA 17740

Listing 1 *(continued)*

```
SCR # 84
  0 ( ESCAPE KEY PROCESSOR )
  1
  2 : ESC                              ( --- )
  3
  4    KEY DOCASE                      ( Do case on the next key )
  5
  6     83 CASE CHANGE-SIDE ENDCASE ( ESC S - change video side )
  7     78 CASE ENEXT       ENDCASE ( ESC N - edit next screen  )
  8     76 CASE ELAST       ENDCASE ( ESC L - edit last screen  )
  9     69 CASE EXIT        ENDCASE ( ESC E - exit editor )
 10     65 CASE EABORT      ENDCASE ( ESC A - abort edit session )
 11     80 CASE PUT-LINE    ENDCASE ( ESC P - put down deleted line
 12
 13     BEEP ENDCASES ;             ( else signal key error )
 14
 15 -->

SCR # 85
  0 ( SCREEN EDITOR DEFINITION )
  1
  2 : EDIT                             ( screen# --- )
  3   SCR !                            ( Save EDIT screen number )
  4   EREAD DISPLAY                    ( Read and display screen )
  5   0 STOP !                         ( Clear stop flag to false )
  6
  7   BEGIN                            ( Begin editing )
  8    KEY DOCASE                      ( Get a key and process )
  9      28 CASE UP-CURSOR    ENDCASE ( ATARI up arrow )
 10      29 CASE DOWN-CURSOR  ENDCASE ( ATARI down arrow )
 11      31 CASE RIGHT-CURSOR ENDCASE ( ATARI right arrow )
 12      30 CASE LEFT-CURSOR  ENDCASE ( ATARI left arrow )
 13     155 CASE RETURN       ENDCASE ( ATARI return key )
 14
 15 -->

SCR # 86
  0 ( SCREEN EDITOR DEFINITION )
  1
  2    255 CASE INSERT       ENDCASE ( ATARI CTRL/INSERT key )
  3    254 CASE DELETE       ENDCASE ( ATARI CTRL/DELETE key )
  4    157 CASE INSERT-LINE  ENDCASE ( ATARI SHIFT/INSERT key )
  5    156 CASE DELETE-LINE  ENDCASE ( ATARI SHIFT/DELETE key )
  6    126 CASE DEL          ENDCASE ( ATARI DELETE key )
  7    127 CASE TAB          ENDCASE ( ATARI TAB key )
  8    125 CASE CLEAR-SCR    ENDCASE ( ATARI CTRL/CLEAR key )
  9     27 CASE ESC          ENDCASE ( Enter escape key function )
 10    DUP ADDKEY  ENDCASES          ( else addkey to text )
 11    STOP @                        ( Check if stop flag set )
 12   UNTIL                          ( Edit until stop is true )
 13
 14   XGR ;                          ( Clear screen, goto FORTH )
 15 ;S
```

control keys (graphics) available for FORTH screen text.

As a final note, most versions of fig-FORTH do not include a CASE statement. However, unlike languages such as BASIC or FORTRAN, FORTH may be extended to include new control structures. EDIT uses a set of CASE words defined by Dr. C.E. Eaker, orginally written for a 6800 FORTH system (*FORTH Dimension*, Volume II, Number 3, pp. 37-40). Only the word names were changed when using this CASE statement. The actual definitions were compiled and executed the first time. Applications written in high-level FORTH (no code words) can usually be transported between FORTH systems, regardless of the processor type.

Conclusion

EDIT is my first large application with the APX fig-FORTH implementation. While EDIT probably is not as efficient as possible, it was written and debugged in less than 15 hours. Any language that allows a large application to be rapidly and logically implemented during the learning process certainly deserves attention! Many well-designed concepts are at work in FORTH and all programmers should consider FORTH as an alternative to BASIC.

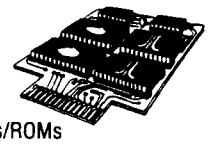
The author may be contacted at 7659 West Fremont Ave., Littleton, CO 80123.

MICRO

Chameleon™

HYPERCARTRIDGE™
for ATARI® 400/800

16K
$39



w/o EPROMs/ROMs

FOR SOFTWARE DEVELOPERS AND HOBBYISTS!

- extend memory of 16K RAM and 32K RAM computers
- create 16K cartridges easily with an EPROM programmer
- combine ATARI® BASIC ROMs with your own subroutines on ROM/EPROM
- eliminate need for disk drive and extra RAM for lengthy programs

CONFIGURATIONS:
#1 Any combination of 4 2532 EPROMs/2332 ROMs
#2 Two ATARI ROMs and two 2532's (or 2332's)
SPECIFY WITH ORDER

Also order:
2532 4K EPROMs $7.50 each
with cartridge order only

****

CHAMELEON COMPUTING™
Dept. of Physics & Astronomy
Box 119-M
Dickinson College
Carlisle, PA 17013

(717) 245-1717

****

Please add:
$1.50 shipping/handling
PA residents add 6% sales tax
CHECK, MC, VISA
Quantity discounts available

APPLE Pascal Hi-Res Screen Dump

by Robert D. Walker

A Pascal procedure to dump the high-resolution graphics screen to your printer.

SCREENDUMP
requires:
Pascal
Epson with Graftrax

Many machine-language subroutines have been written to dump the Apple high-resolution graphics to the Epson MX-80 printer. I have not found, however, any subroutine specifically written for Apple Pascal. If you own an Epson MX-80 (with Graftrax) and want a hard copy of Turtlegraphics, the following Pascal procedure should prove handy.

The procedure in listing 1 takes advantage of Pascal's ability to declare variant records. The type WIRES associates eight boolean variables with eight bits in a byte. In fact, these eight boolean variables occupy the same memory location as the byte. Each boolean variable represents one wire on the printhead. If the boolean variable is true, the printing wire is fired, otherwise it is not.

I encountered one problem during the testing of this procedure. The intrinsic WRITE procedure does not allow all character codes to be passed to the printer. This problem is circumvented by using the low-level procedure UNITWRITE (see pp. 41-42 of the *Apple Pascal Language Reference Manual*).

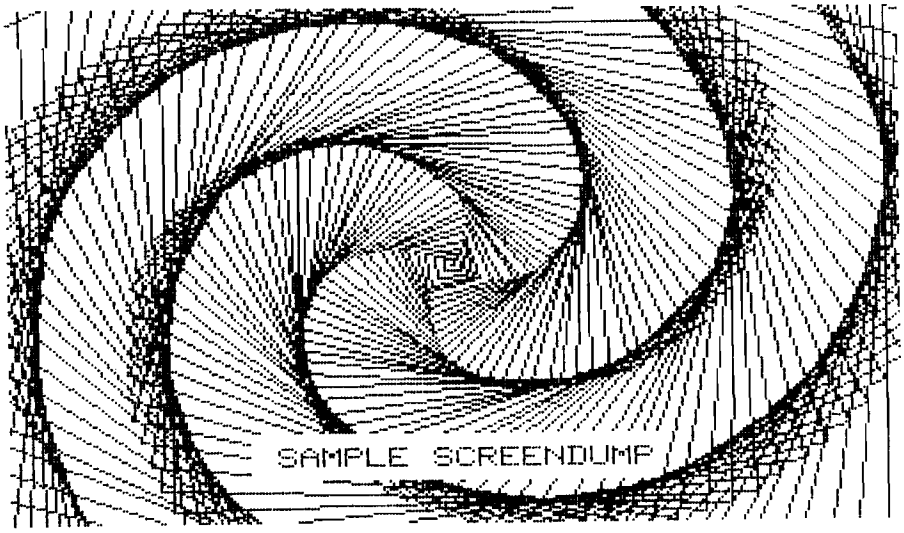
The SCREENDUMP procedure has the following form: SCREENDUMP (LEFT, RIGHT, BOTTOM, TOP, LMARGIN)
LEFT = left X position to be printed
RIGHT = right X position to be printed
BOTTOM = bottom Y position to be printed
TOP = top Y position to be printed
LMARGIN = number of spaces in left margin
(*Note:* both Y values are rounded to the next lower integer evenly divisible by eight.)

Using SCREENDUMP is simple. I have included a sample printout and a short program that demonstrates its use. The procedure is easy to use and there is no need for error checking. It is the programmer's responsibility to ensure that all parameters are within the proper range.

I have used this procedure in many programs involving Turtlegraphics. Because of the popularity of the Apple II and the Epson MX-80, this procedure should find a place in many subroutine libraries.

You may contact the author at 2850 Delk Rd., Apt. 2B, Marietta, GA 30067.

***Figure 1:* Sample Ouptut From SCREENDUMP**



Listing 1: SCREENDUMP

```
PROGRAM SAMPLEUSEOFSCREENDUMP;  (* SAMPLE PROGRAM USING SCREENDUMP *)

USES TURTLEGRAPHICS;

VAR DISTINC:  INTEGER;



PROCEDURE SCREENDUMP(LEFT, RIGHT, BOTTOM, TOP, LMARGIN:  INTEGER);
{**************************************************************************}
{**                                                                      **}
{**  THIS PROCEDURE DUMPS THE APPLE HIRES SCREEN TO THE EPSON MX-80      **}
{**  PRINTER EQUIPPED WITH GRAFTRAX.  TURTLEGRAPHICS PROCEDURES AND      **}
{**  FUNCTIONS ARE USED.                                                 **}
{**                                                                      **}
{**  R. WALKER  7/82                                                     **}
{**  MARIETTA, GA                                                        **}
{**                                                                      **}
{**************************************************************************}

TYPE  BYTE=0..255;

      (* USE VARIANT RECORD TO ASSOCIATE PRINTING WIRES WITH BYTE *)
      WIRES=PACKED RECORD CASE BOOLEAN OF
             TRUE:   (BO:  PACKED ARRAY[0..7] OF BOOLEAN);
             FALSE:  (BY:  BYTE)
           END;

VAR  I,X,YCOARSE,YFINE:  INTEGER;
     BITIMAGE:  PACKED ARRAY[0..200] OF BYTE;
     PRINTCODE:  PACKED ARRAY[1..4] OF BYTE;
     W:  WIRES;

BEGIN  (* SCREENDUMP *)

  (* SET LINE SPACING TO 24/216" *)
  PRINTCODE[1]:=27;  PRINTCODE[2]:=51;  PRINTCODE[3]:=24;
  UNITWRITE(6,PRINTCODE[1],3,0,12);


  (* PRINT SCREEN *)
  FOR YCOARSE:= (TOP DIV 8) DOWNTO (BOTTOM DIV 8) DO
    BEGIN

      (* MAKE LEFT MARGIN *)
      PRINTCODE[1]:=32;
      FOR I:=1 TO LMARGIN DO UNITWRITE(6,PRINTCODE[1],1,0,12);

      (* ASSEMBLE ONE LINE OF BIT IMAGES *)
      FOR X:=LEFT TO RIGHT DO
        BEGIN
          FOR YFINE:=0 TO 7 DO W.BO[YFINE]:=SCREENBIT(X,YCOARSE*8+YFINE);
          BITIMAGE[X]:=W.BY
        END;

      (* TELL PRINTER HOW MANY DOTS IN LINE *)
      PRINTCODE[1]:=27;  PRINTCODE[2]:=75;
      PRINTCODE[3]:=(RIGHT-LEFT+1) MOD 256;
      PRINTCODE[4]:=(RIGHT-LEFT+1) DIV 256;
      UNITWRITE(6,PRINTCODE[1],4,0,12);

      (* SEND ONE LINE OF BIT IMAGES TO PRINTER *)
      BITIMAGE[RIGHT+1]:=10;  (* LINEFEED *)
      UNITWRITE(6,BITIMAGE[LEFT],(RIGHT-LEFT+2),0,12)
    END;

  (* RESET PRINTER *)
  PRINTCODE[1]:=27;  PRINTCODE[2]:=64;
  UNITWRITE(6,PRINTCODE[1],2,0,12)

END;  (* SCREENDUMP *)
```

Listing 1: SCREENDUMP *(continued)*

```
BEGIN (* MAIN PROGRAM *)

  (* DRAW PICTURE *)
  INITTURTLE;
  PENCOLOR(WHITE);
  DISTINC:=1;
  REPEAT
    MOVE(DISTINC);
    TURN(91);
    DISTINC:=DISTINC+1
  UNTIL DISTINC=300;
  VIEWPORT(76,203,16,32);
  FILLSCREEN(BLACK);
  PENCOLOR(NONE);
  MOVETO(84,20);
  WSTRING('SAMPLE SCREENDUMP');

  (* CALL SCREENDUMP *)
  SCREENDUMP(0,279,0,191,17)

END.
```

MICRO

FEATURES: Efficient 30kHz switching frequency • Four Models satisfy most applications • Years of trouble-free service • Each side AC line fuse protected • Tele-Tale LED "Pwr On" Panel Indicator • Three separate voltage outputs • Metal enclosure provides physical and EMI protection • For experimental use or permanent power source • Soft start feature protects critical circuits • Parallel operation acceptable for higher current needs • Push-in terminals, accept wire or test lead • Light-weight, easy to use • AC line cord permanently attached • Most reliable power source for a variety of uses and applications • 48 hour burn-in assures MTBF of 3½ years, reasonably priced at $1.90/watt • Full one year guarantee • 2-tone anodized case • Custom volt/current outputs on special order • Input surge protection • Automatic short circuit protection + restoration • UL recognized components • Handy Service Aid.

SPECIFICATIONS: Input: 90-132VAC, 47-440Hz • Dual AC Input Fuses • Line Regulation: ±0.1% Max. for 10% input change • Load Regulation: ±0.2% Max. on #1 Output • Ripple Noise: Typ. 1% PP Max. • Over Voltage Protection • Reverse Polarity Protection • Compact, only 7½" x 4" x 2¾" • Fast load transient response • 5 volt adj. ±10% • DC Output: 42 Watts continuous • 70% Efficiency

SCHOOLS—LABS: QUANTITY PRICING ON REQUEST

Lcom inc 1545 Osgood St. Unit 11H, No. Andover, MA 01845

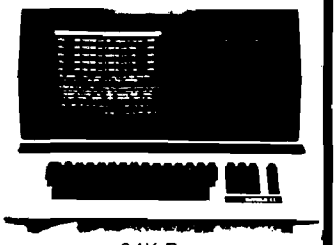
Name (Please print)_____
Address_____
City_____ State_____ Zip_____

Qty.	Model	Output #1	Output #2	Output #3	Total
	PS–1	5V-6A	+12V-0.5A	–12V-0.5A	
	PS–2	5V-6A	+15V-0.4A	–15V-0.4A	
	PS–3	5V-6A	+12V-0.5A	–5V-1A	
	PS–4	5V-3A	+24V-0.6A	–24V-0.6A	
	Information on other switcher models				NC

ORDER INFORMATION	
Order First Unit—$99.50	
Second Unit—$79.60	
OFFER EXPIRES March 31, 1983	
Sub-Total	
Mass. res add 5% Tax	
Shipping & Handling	3.50
TOTAL	

Charge to: ☐ MasterCard ☐ Visa ☐ American Express ☐ Check/Money Order
Card #_____ Exp. Date_____
Signature_____
PHONE ORDERS: CALL (617)682-6936 FOR PROMPT SERVICE

EAGLE

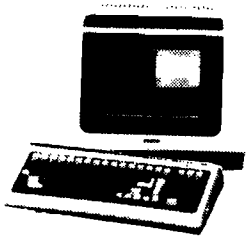


64K Ram
780 KB Disk Storage
Word Processing, Ultracalc CP/M
C-Basic Software
Smith Corona TP 1
Letter Quality Printer

$2995.00

Retail Value $4895.00

EAGLE 1600... CALL



TELEVIDEO TERMINALS

910	$579.00
912C	$699.00
920C	$749.00
925C	$749.00
950	$950.00

TELEVIDEO COMPUTERS

800A	$1319.00
802	$2649.00
802H	$4695.00
806	$5495.00
816	$9495.00
803	CALL
1603	CALL

MONITORS

AMDEK

100 B & W	$74.95
300G	$169.00
300A	$179.00
Color I	$339.00
Color II	$699.00
Color II A	$799.00
Color III	$399.00
Color IV	CALL

BMC

12" Green	$79.99
13" Color 1401 (Mid Res.)	$369.00
9191U 13"	$329.00

ZENITH

ZVM 121	$99.00

SHARP

Sharp 13" Color TV	$275.00

PANASONIC

TR-120MIP (High Res. Green)	$159.00
CT-160 Dual Mode Color	$299.00

HEWLETT PACKARD

41CV

$209

HP 41C	$149.00
HP 10C	$69.00
HP 11C	$79.00
HP 12C	$114.00
HP 15C	$109.00
NEW 16C	$114.00

PERIPHERALS

HP41 Card Reader	$144.00
HPIL Module	$99.00
HPIL Cassette	$449.00
HPIL Printer	$419.00
Quad Memory Module	$64.00
Time Module	$64.00
Extended Function Module	$64.00

NEC

COMPUTERS

8001A	$729.00
8031	$729.00
8012	$549.00

PRINTERS

8023	$499.00
7710/7730	$2399.00
3510/3530	$1599.00

MONITORS

JB-1260	$129.00
JB-1201	$159.00
JC-1201	$319.00
JC-1203	$729.00

TIMEX SINCLAIR 1000

$89.99

16K Memory Module	$44.95
Vu-Calc	$17.95
Super Math	$12.95
Check Book Manager	$13.95
The Organizer	$14.95
The Budgeter	$13.95
Stock Option	$14.95
Loan & Mortgage Amortizer	$12.95

SHARP PC-1500 POCKET COMPUTER

$209

CE 150 Printer, Plotter and Cass. Interface Unit... $172.00
CE 152 Cass. Recorder... $69.00
CE 155 8K Ram Expansion Module..... $94.00

HP●85 $1969

HP 125	$1999.00
HP 85 16K Memory Module	$169.00
5¼" Dual Floppy Disk	$1799.00
Hard Disk w/Floppy	$4349.00
Hard Disk	$3549.00
"Sweet Lips" Printer	$1219.00
80 Column Printer	$649.00

IBM®

NEC 3550 PRINTER... $2099

PERCOM DRIVES

5¼" 160K Disk Drive	$249.00
5¼" 320K Disk Drive	$299.00

AMDEK

310A Amber Monitor	$179.00
310G	$179.00
Amdisk (3¼" Drive)	$729.00
DXY Plotter	$759.00
Color II	$699.00

SOFTWARE

I.U.S. Easywriter II	$249.00
I.U.S. Easyspeller	$129.00
Peach Package (GL/AP/AR)	$419.00

PROFESSIONAL SOFTWARE

IBM/PC Word Processing	$319.00

PRINTERS

SMITH CORONA

TP 1	$599.00

C. ITOH (TEC)

Starwriter(F10-40CPS)	$1399.00
Printmaster(F10-55CPS)	$1749.00
Prowriter 80 Col.(Parallel)	$499.00
Prowriter 80 Col.(Serial)	$629.00
Prowriter 2 (132 Col.)	$799.00

OKIDATA

82A	$429.00
83A	$659.00
84 (Parallel)	$1079.00
84 (Serial)	$1199.00

IDS

MicroPrism	$649.00
132 (Fully Configured)	$1599.00
80 (Fully Configured)	$1399.00

Call for other configurations.

STAR

Gemini 10	$379.00

DAISYWRITER

Letter Quality	1049.00

DIABLO

620	$1179.00
630	$1849.00

MODEMS

HAYES

Smart	$239.00
Smart 1200 (1200 Baud)	$549.00
Chronograph	$199.00
Micromodem II (with Term)	$309.00
Micromodem 100	$309.00

NOVATION

Cat	$144.00
D-Cat	$159.00
212 Auto Cat	$589.00
Apple Cat II	$279.00
212 Apple Cat II	$609.00

CALL for Price and Availability on New **NOVATION** Cat 103, 103/212 and J-Cat.

ANCHOR

Mark I (RS-232)	$79.00
Mark II (Atari)	79.00
Mark III (TI-99)	109.00
Mark IV (CBM/PET)	$125.00
Mark V (OSBORNE)	$95.00
Mark VI (IBM-PC)	$179.00
Mark VII (Auto Answer Call)	$119.00
TRS -80 Color Computer	$99.00
9 Volt Power Supply	$9.00

commodore

8032	$1039.00
CBM 64	CALL
4032	$749.00
8096 Upgrade Kit	$369.00
Super Pet	$1499.00
2031	$469.00
8250 Dbl.Sided Disk Drive	$1899.00
D9060 5 Meg. Hard Disk	$2399.00
D9060 7.5 Meg. Hard Disk	$2699.00
8050	$1299.00
4040	$969.00
8300 (Letter Quality)	$1549.00
8023	$599.00
4022	$399.00
New Z-Ram, Adds CP/M & 64K	$549.00
The Manager	$209.00
Magis	CALL
Word Pro 5 Plus	$319.00
Word Pro 4 Plus	$299.00
Word Pro 3 Plus	$199.00
The Administrator	$379.00
Info Pro Plus	$219.00
Power	$79.00
CBM 8032 Dust Cover	$14.99
CBM 8050/4040 Dust Cover	$10.99

computer mail order east

800-233-8950

IN PA. CALL (717)327-9575, 477 E.THIRD ST., WILLIAMSPORT, PA. 17701

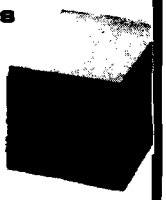
In stock items shipped same day you call. No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the continental United States with no waiting period for certified checks or money orders. Add 3% (minimum $3.00) shipping and handling on all C.O.D. and Credit Card orders. NV and PA residents add sales tax. All items subject to availability and price change. **NOTE:** We stock manufacturer's and third party software for most all computers on the market! CALL TODAY FOR OUR NEW CATALOGUE.

FRANKLIN

ACE 1000
ACE 10 with Controller Card
ACE Writer Word Processor
CALL...
FOR SYSTEM PRICE!
ACE 1200..... CALL

PERCOM

DISK DRIVES FOR ATARI



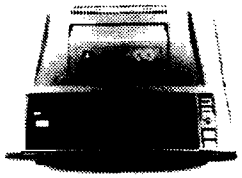
AT 88-S1....$399.00
AT 88-A1....$289.00
RFD 40-S1...$539.00
RFD 40-A1...$329.00
RFD 40-S2...$869.00
RFD 44-S1...$659.00
RFD 44-S2...$999.00

RANA DISK DRIVES

Call for price and availability on the new Rana Disk Drives for The Apple and Franklin Computer Systems.

HOMECOMPUTERS

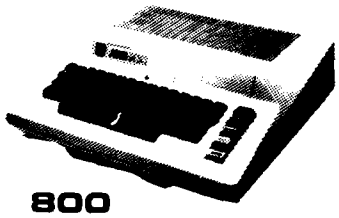
ATARI®



400

16K.....$199
32K.....$274*
48K.....$359*
*Non-Atari Ram

410 Recorder	$74.00
810 Disk Drive	$429.00
822 Printer	$269.00
825 Printer	$589.00
830 Modem	$159.00
820 Printer	$259.00
850 Interface	$169.00
CX40 Joy Sticks (pair)	$18.00
CX853 Atari 16K Ram	$77.95



800

48K.....$499
New low price effective January 1, 1983.

Call for Price and Availability of the NEW
64K ATARI 1200

Axlon Ramdisk (128K)	$429.95
Intec 48K Board	$159.00
Intec 32K Board	$74.00
One Year Extended Warranty	$70.00
CX481 Entertainer Package	$69.00
CX482 Educator Package	$130.00
CX483 Programmer Package	$54.00
CX484 Communicator Package	$344.00

VISICORP

for Apple, IBM & Franklin

Visidex	$189.00
Visifile	$189.00
Visiplot	$159.00
Visiterm	$89.00
Visitrend/Plot	$229.00
VisiSchedule	$229.00
Desktop Plan	$189.00
Visicalc (Apple II, Atari, CBM, IBM)	$179.00

Visicorp prices for IBM may vary slightly.

CONTINENTAL

Home Accnt. (Apple/Franklin)	$59.00
Home Accountant (IBM)	$119.00
1st Class Mail (Apple/Franklin)	$59.00

SIRIUS

Free Fall	$24.00
Beer Run	$24.00
Snake Byte	$24.00
Space Eggs	$24.00
Sneakers	$24.00
Bandits	$28.00

BRODERBUND

Apple Panic	$23.00
David's Magic	$27.00
Star Blazer	$25.00
Arcade Machine	$34.00
Choplifter	$27.00
Serpentine	$27.00

INFOCOM

Deadline	$35.00
Star Cross	$29.00
Zork I	$29.00
Zork II or III	$29.00

MPC

Bubdisk (126K Ram)	$719.00

AXLON

Ram Disk (Apple/Franklin)	CALL

μ-SCI

MICRO-SCI DISK DRIVES FOR APPLE & FRANKLIN

A2	$299.00
A40	$349.00
A70	$459.00
C2 Controller	$79.00
C47 Controller	$89.00

FLOPPY DISKS

MAXELL

MD I (Box of 10)	$32.00
MD II (Box of 10)	$44.00
FD I (8")	$40.00
FD II (8" DD)	$50.00

VERBATUM

5¼" SS DD	$26.00
5¼" DS DD	$36.00

ELEPHANT

5¼" SS SD	$19.99

VIC 20

$179.

VIC 20 Dust Cover	$9.99
VIC 1530 Datassette	$69.00
VIC 1540 Disk Drive	$339.00
VIC 1541 (64K Disk Drive)	CALL
VIC 1525 Graphic Printer	$339.00
VIC 1210 3K Mem. Exp.	$32.00
VIC 1110 8K Mem. Exp.	$53.00
VIC 1111 16K Mem. Exp.	$94.00
VIC 1011 RS232C Term. Interface	$43.00
VIC 1112 IEEE-488 Interface	$86.00
VIC 1211 Super Expander	$53.00
VIC Mother Board	$99.00

Call for Price on

VIC 64

Peripherals and Software.

PROFESSIONAL SOFTWARE
Word Processing for VIC 64....$79.95

SOFTWARE FOR ATARI

ATARI

Pac-Man	$33.00
Centipede	$33.00
Caverns of Mars	$32.00
Asteroids	$29.00
Missile Command	$29.00
Star Raiders	$35.00
Galaxian	$33.00
Defender	$33.00

ON-LINE

Jawbreaker	$27.00
Softporn	$27.00
Wizard and the Princess	$29.00
The Next Step	$34.00
Mission Asteroid	$22.00
Mouskattack	$31.00
Frogger	$31.00
Cross Fire (ROM)	$36.00

SYNAPSE

File Manager 800 †	$69.00
Chicken	$26.00
Dodge Racer	$26.00
Synassembler	$30.00
Page 6	$19.00
Shamus	$26.00
Protector	$26.00
Nautilus	$26.00
Slime	$26.00
Disk Manager	$24.00

DATASOFT

Pacific Coast Highway	$25.00
Canyon Climber	$25.00
Tumble Bugs	$25.00
Shooting Arcade	$25.00
Clowns and Balloons	$25.00
Graphic Master	$30.00
Graphic Generator	$13.00
Micro Painter	$25.00
Text Wizard	$79.00
Spell Wizard	$84.00
Bishop's Square	$25.00
Sands of Egypt	$25.00

APX

Text Formatter	$18.50
Family Budgeter	$18.50
Eastern Front	$24.00
Family Cash	$18.50
Jukebox	$13.00
Downhill	$18.50
Outlaw	$18.50
Holy Grail	$24.00
Player Piano	$18.50
Keyboard Piano	$18.50
Number Blast	$13.00
Frogmaster	$18.50
747 Land Simulator	$18.50
Word Processor	$40.00

EPYX

Crush, Crumble & Chomp	$24.00
Crypt of the Undead	$24.00
Curse of Ra	$16.00
Datestones & Ryn	$16.00
Invasion Orion	$19.00
King Arthur's Heir	$24.00
Morloc's Tower	$16.00
Rescue at Rigel	$24.00
Ricochet	$16.00
Star Warrior	$29.00
Temple of Asphai	$29.00
Upper Reaches of Apshai	$16.00

CBS

K-razy Shoot Out	$32.00
K-razy Kritters	$32.00
K-razy Antics	$32.00
K-star Patrol	$32.00

STICK STAND $6.99

Arcade Action from your ATARI or VIC Joy Stick

computer mail order west

800-648-3311

IN NV. CALL (702)588-5654, P.O. BOX 6688, STATELINE, NV. 89449

INTERNATIONAL ORDERS: All shipments outside continental United States must be pre-paid by certified check only! Include 3%(minimum $3.00) shipping and handling.
EDUCATIONAL DISCOUNTS: Additional discounts are available from both Computer Mail Order locations to qualified Educational Institutions.

An Introduction to FORTH

by Ronald W. Anderson

The author gives a brief introduction to the FORTH language, including a discussion of Reverse Polish Notation, word definitions, and stack manipulation.

FORTH was the result of the old adage that "necessity is the mother of invention." Charles Moore developed FORTH as a tool to help him program computers more quickly than he could with an Assembler.

If you have used a Hewlett Packard calculator, you are familiar with Reverse Polish Notation. FORTH works with this notation exclusively. Reverse Polish works well with a stack structure. HP used it to simplify the use of their calculators. The difference between Reverse Polish and ordinary algebraic notation may be seen in the way you key a simple problem into calculators that use these two notations respectively.

Algebraic

2 + 2 =
2 × 3 + 4 × 5 =
(2+3) × (4+5) =

Reverse Polish

2 (enter) 2 +
2 (enter) 3 × 4 (enter) 5 × +
2 (enter) 3 + 4 (enter) 5 + ×

The second and third problems above are done in exactly the same way; in algebraic notation, parentheses are necessary for one case but not the other, since multiplication takes precedence over addition. When using a Reverse Polish calculator, ENTER puts the first argument on the stack. The operator + or ×, for example, puts the second number on the stack and operates on the top two numbers, removing the two numbers that were there and leaving the result on top of the stack. (In the case of the calculator, the top item on the stack is always displayed.) With this notation, you can put a group of intermediate results on the stack and then perform the final operations. Though the idea might seem a bit strange at first, most HP users will testify that the operations may be performed with little or no thought.

Several years ago, *Consumer's Report* did a review of all the calculators available. They downgraded the HP severely because of the "strange notation." In a note a few issues later, they did a reverse. It seems that everyone who used the calculators eventually wound up looking for the HP because it was easier to get the correct answer on it.

FORTH, as you may have realized by now, relies heavily on a stack for all calculations. All of FORTH's instructions in some way manipulate the information on the stack. FORTH instructions are called "words." A word is defined by a "colon definition." A word may have *any* combination of ASCII characters as its name. "." is a FORTH word meaning the same thing as PRINT in BASIC. If you typed in the instructions:

```
2 2 * . (return)
```

FORTH would respond with:

```
4 OK
```

All FORTH words or instructions must be separated by spaces. When FORTH sees a literal number, it automatically puts it on the stack, so an equivalent of the (enter) from the calculator is not necessary. (The space after the number tells FORTH that the number is complete.) Of course the "*" means multiply, and the "." means to print the result. Printing a result removes it from the stack. Assuming the stack was empty at the start of the above sequence of instructions, it would be empty at the end. "OK" is FORTH's analog of READY in BASIC. Shown in figure 1 are the contents of the stack, as each item in the line above is encountered.

Figure 1

	2	**3**	*	.
TOP	2	3	6	(empty)
OF → (empty)		2		
STACK				

Now, let's define a word:

```
: SQUARE DUP * ;
```

The word defined with a colon definition is SQUARE. It will square the value on the stack. Now, if you type 2 SQUARE . (return) FORTH will put 2 on the stack. DUP is a FORTH word that will push a DUPlicate of the top item onto the stack. "*" multiplies the top two items on the stack and leaves the result on top. 2 SQUARE . will therefore result in the value 4 being printed to the terminal. Now, of course, if you typed '5 SQUARE .' you would get 25 on the terminal. Further, the result of SQUARE does not need to be printed out. It could just as well be left on the stack for use by another calculation. You could type:

```
3 SQUARE 4 SQUARE + . (RETURN)
```

The result would be 9 + 16 or: 25 OK

The definition of SQUARE is comprised of just two other words, DUP and *. The semicolon terminates the word definition.

Figure 2 shows a colon definition of a working square root function. First, a couple of variables are declared. They hold the number for which the square root is to be found, and for an intermediate GUESS or trial square root. The algorithm, called Newton's method, divides the number by a guess, and averages the result with the guess to make a new guess. The process is repeated until the new guess and the old guess are either equal, or differ by 1. (Remember, this is integer arithmetic. The result for some numbers will alternate between two numbers that differ by 1; for others it will reach a constant value.)

SQRT expects the value of the number to be on the stack when SQRT is called. The number will be used several times in successive passes through the loop, so it is immediately stored in the variable NUMBER. Naming a variable places its address on the stack. The word '!' makes FORTH use the top item on the stack as a pointer for a place to store the second item on the stack. Remember that you are usually dealing with 16-bit words. 2 GUESS ! puts 2 on the stack and stores it in the variable GUESS. BEGIN signals the start of a loop that ends at UNTIL. Within the loop, NUMBER and GUESS are fetched. @ has the reverse effect of !; it uses the top item on the stack as a pointer to a variable, and replaces the pointer value with the value of the variable on the stack.

Figure 2

```
0 VARIABLE GUESS
0 VARIABLE NUMBER

: SQRT
  NUMBER !     2 GUESS !
  BEGIN NUMBER @ GUESS @ /
  GUESS @ + 2 /  DUP
  GUESS @ SWAP  GUESS !
  - ABS  2 < UNTIL
  GUESS @ ;
```

Next, the word '/' divides NUMBER by GUESS; the result remains on the stack. Now GUESS is fetched again and added to the result. 2 / divides the sum by 2 and you have averaged the result of the divide with the original guess, so the new guess is now on the stack. DUP duplicates the new guess on top of the stack, and GUESS @ puts the old one on top. Now you want to save the new guess in GUESS, but it is second on the stack, so use SWAP to get it on top and then GUESS ! to put it in GUESS. Now you have the new guess and the old guess as the two top items on the stack so subtract and take the absolute value of the difference (– ABS). You must set up a comparison that will leave FALSE (0) on the stack until you want to exit the loop. 2 < UNTIL compares the value on the stack with 2 and leaves TRUE when the value is less than 2 (0 or 1). At that point the loop is done and you simply pull the last guess as the result and return with the result on the stack.

It is my understanding that an avid FORTH fan frowns on the use of variables if it can be avoided. He would probably figure out a way to keep both GUESS and NUMBER on the stack (as nameless values) and manipulate the values with DUP, ROT, and OVER, words that move the top values around in various ways. I believe such code,

SOFTWARE FOR THE HARDCORE

THE PROFESSIONAL'S CHOICE
FORTH — A Tool for Craftsmen!

It has been said that if Chippendale had made programs he would have used FORTH as his tool. If you want to learn how to program, use a teaching language—PASCAL or BASIC. If you know how to program, use a language designed for craftsmen—FORTH.

FORTH Systems

For all FLEX systems: 6800 & 6809. Specify 5" or 8" diskette and hardware configuration. For standalone versions, write or call.

** tFORTH—extended fig-FORTH (1 disk) $100 ($15)
** tFORTH + —extended more! (3 5" or 2 8" disks) $250 ($25)
tFORTH + includes 2nd screen editor, assembler, extended data types and utility vocabularies, GOING FORTH CAI course on FORTH, games, and debugging aids.

TRS-80 COLORFORTH — 10K ROM Pack

Full screen editor. Will work on 4K, 16K, or 32K systems $110 ($20). Disk versions available.

Applications Programs

** firmFORTH 6809 tFORTH + only $350 ($10)
For target compilations to rommable code. Deletes unused code and unneeded dictionary heads. Requires tFORTH +.
** TINY PASCAL compiler in FORTH. 6800/09 $75 ($20)
** FORTH PROGRAMMING AIDS: Extensive debugging, decompiling, and program analysis tools. $150 ($10)

Manuals alone, price in ( ). Add $5/system for shipping. $12 for foreign air.

Talbot Microsystems
1927 Curtis Ave., Redondo Beach, CA 90278
(213) 376-9941

(TM) tFORTH, COLORFORTH and firmFORTH are trademarks of Talbot Microsystems.
(TM) FLEX is a trademark of Technical Systems Consultants.

FORTH-79

Ver. 2 For your APPLE II/II+

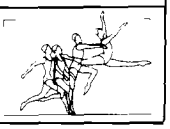
The complete professional software system, that meets ALL provisions of the FORTH–79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH–79 with the FORTH you are now using, or plan to buy!

FEATURES	OURS	OTHERS
79-Standard system gives source portability.	YES	
Professionally written tutorial & user manual	200 PG.	
Screen editor with user-definable controls.	YES	
Macro-assembler with local labels.	YES	
Virtual memory.	YES	
Both 13 & 16-sector format.	YES	
Multiple disk drives.	YES	
Double-number Standard & String extensions.	YES	
Upper/lower case keyboard input.	YES	
LO-Res graphics.	YES	
80 column display capability	YES	
Z-80 CP/M Ver. 2.x & Northstar also available	YES	
Affordable!	$99.95	
Low cost enhancement option:		
Hi-Res turtle-graphics.	YES	
Floating-point mathematics. Powerful package with own manual, 50 functions in all, AM9511 compatible.	YES	

FORTH–79 V.2 (requires 48K & 1 disk drive)	$ 99.95
ENHANCEMENT PACKAGE FOR V.2 Floating point & Hi-Res turtle-graphics	$ 49.95
COMBINATION PACKAGE	$139.95

(CA res. add 6% tax: COD accepted)

MicroMotion
12077 Wilshire Blvd. #.506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.



though perhaps a bit more efficient, is too hard to follow. The use of variables with fetch and store instructions makes the program much more readable.

Now let's define a third word, HYPOTENUSE, that will put the first two together.

```
: HYPOTENUSE
    SQUARE SWAP SQUARE + SQRT ;
```

HYPOTENUSE is called with the length of the two sides of a right triangle on the stack. It returns the length of the hypotenuse.

```
  3   4 HYPOTENUSE .   5 OK
  5  12 HYPOTENUSE .  13 OK
100 100 HYPOTENUSE . 141 OK
```

The above example shows how to first define low-level words and then use them to define higher-level words. A FORTH program is built like this, starting with low-level word definitions, and continuing with higher- and higher-level words, until very few words can be combined to accomplish a task.

The trick of writing programs in FORTH is simply to keep track of what is on the stack at every point in the program. This is facilitated by keeping word definitions short. A bit of code that leaves a data value on the stack somewhere in the program may do no harm, but if it is in a loop that repeats many times, it will eventually cause the stack to overflow and crash the program. Code that removes more from the stack than it puts on causes a STACK UNDERFLOW error and most FORTH implementations will tell you so.

Actually, the previous discussion is a bit of an oversimplification, since FORTH actually uses two stacks — the parameter stack and the return stack. The return stack is used to keep track of the program flow from colon definition to colon definition, and it is usually transparent to the user. Advanced techniques may involve using words to manipulate the return stack, as well.

FORTH has a core of standard words for mathematical operators, data-handling words, and decision making. One of the differences among the FORTH implementations is how many core words are implemented in assembler, as opposed to word definitions in FORTH. Assembler-defined words generally run faster than the FORTH-defined words. Also, the main part of FORTH will be smaller if more of the core is implemented in assembler.

The normal math functions in FORTH are based on 16-bit integer arithmetic, so numbers have the range of 32767 to −32768. The implementations discussed in the companion article (pg. 62) have additional standard FORTH words defined that allow manipulation of larger integer numbers, such as D* for double precision multiply, etc. You can write a floating-point math package in FORTH if it is required for your applications. You can tailor the package to your needs, so that you don't have to suffer with the slow calculations of a 12-digit floating-point math if you need only six or seven digits. Such routines have been published and some are commercially available.

The author may be contacted at 3540 Sturbridge Ct., Ann Arbor, MI 48105.

MICRO

Perry Peripherals Repairs KIMs!!

(SYMs and AIMs Too)

- We will Diagnose, Repair, and Completely Test your Single Board Computer
- We Socket all replaced Integrated Circuits
- You receive a 30-day Parts and Labor Warranty
- Your repaired S.B.C. returned via U.P.S. — C.O.D., Cash

Don't delay! Send us your S.B.C. for repair today
Ship To: (Preferably via U.P.S.)

Perry Peripherals
6 Brookhaven Drive
Rocky Point, NY 11778

KIM-1 Replacement Modules

- Exact replacement for MOS/Commodore KIM-1 S.B.C.
- Original KIM-1 firmware — 1K and 4K RAM versions

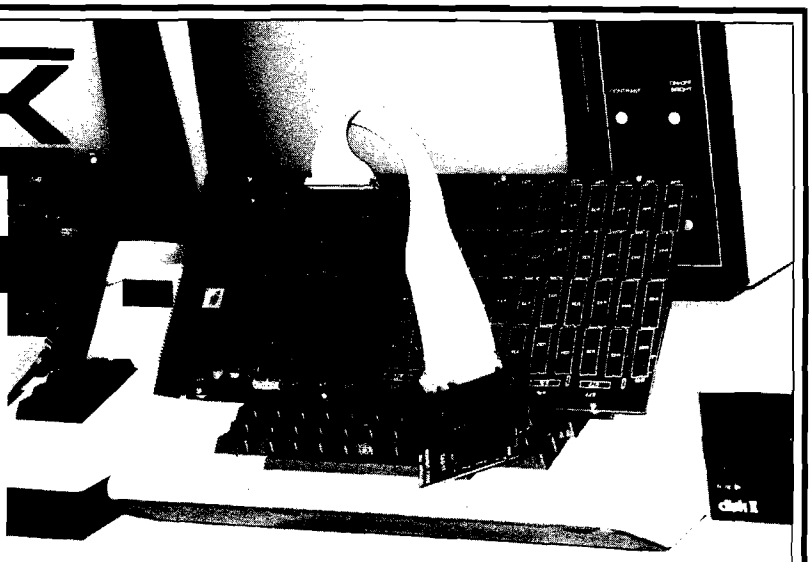
Replacement KIM-1 Keyboards

- Identical to those on early KIMS — SST switch in top right corner
- Easily installed in later model KIMs

Perry Peripherals is an authorized HDE factory service center.

Perry Peripherals carries a full line of the acclaimed HDE expansion components for you KIM, SYM, and AIM, including RAM boards, Disk Systems, and Software like HDE Disk BASIC V1.1. Yes, we also have diskettes. For more information write to: P.O Box 924, Miller Place, NY 11764, or Phone (516) 744-6462.

DTACK



The 68000
DREAM MACHINE

WE (SORT OF) LIED:
Motorola has been promoting its advanced microprocessor chip as a vehicle for large, complex systems **exclusively**. Now, the 68000 does work well as the heart of big, complex systems. But their promotional literature implies that one can **only** build big, complex systems with the 68000, and that is dead wrong (in our opinion). Nevertheless, the public (that's you!) perception of the 68000 follows Motorola's line: **Big** systems. **Complex** systems.

Our boards are **not** complex and not necessarily big (starting at 4K). Our newsletter is subtitled "The Journal of Simple 68000 Systems." But since the public has become conditioned to the 68000 as a vehicle for FORTRAN, UNIX, LISP, PASCAL and SMALLTALK people naturally expect all these with our $595 (starting price) simple attached processor. **Wrong!**

We wrote our last ad to **understate** the software we have available because we wanted to get rid of all those guys who want to run (multi-user, multi-tasking) UNIX on their Apple II and two floppy disks. Running UNIX using two 143K floppies is, well, absurd. The utilities alone require more than 5 megabytes of hard disk.

HERE'S THE TRUTH:
We **do** have some very useful 68000 utility programs. One of these will provide, in conjunction with a suitable BASIC compiler such as PETSPEED (Pet/CBM) or TASC (Apple II), a five to twelve times speedup of your BASIC program. If you have read a serious compiler review, you will have learned that compilers cannot speed up floating point operations (especially transcendentals). Our board, and the utility software we provide, **does** speed up those operations.

Add this line in front of an Applesoft program:

```
5 PRINT CHR$(4);"BLOADUTIL4,A$8600":CALL38383
```

That's all it takes to link our board into Applesoft (assuming you have Applesoft loaded into a 16K RAM card). Now run your program as is for faster number-crunching or compile it to add the benefit of faster "interpretation". Operation with the Pet/CBM is similar.

68000 SOURCE CODE:
For Apple II users only, we provide a nearly full disk of **unprotected** 68000 source code. To use it you will have to have DOS toolkit ($75) and ASSEM68K ($95), both available from third parties. Here's what you get:

1) 68000 source code for our Microsoft compatible floating point package, including LOG, EXP, SQR, SIN, COS, TAN, ATN along with the basic four functions. The code is set up to work either linked into BASIC or with our developmental HALGOL language. 85 sectors.

2) 68000 source code for the PROM monitor. 35 sectors.

3) 68000 source code for a very high speed interactive 3-D graphics demo. 115 sectors.

4) 68000 source code for the HALGOL threaded interpreter. Works with the 68000 floating point package. 56 sectors.

5) 6502 source code for the utilities to link into the BASIC floating point routines and utility and debug code to link into the 68000 PROM monitor. 113 sectors.

The above routines almost fill a standard Apple DOS 3.3 floppy. We provide a second disk (very nearly filled) with various utility and demonstration programs.

SWIFTUS MAXIMUS:
Our last advertisement implied that we sold 8MHz boards to hackers and 12.5MHz boards to businesses. That was sort of true because when that ad was written the 12.5MHz 68000 was a very expensive part (list $332 ea). Motorola has now dropped the price to $111 and we have adjusted our prices accordingly. So now even hackers can afford a 12.5MHz 68000 board. With, we remind you, **absolutely zero wait states.**

'Swiftus maximus'? Do you know of any other microprocessor based product that can do a 32 bit add in 0.48 microseconds?

AN EDUCATIONAL BOARD?
If you want to learn how to program the 68000 at the assembly language level there is no better way than to have one disk full of demonstration programs and another disk full of machine readable (and user-modifiable) 68000 source code.

Those other 'educational boards' have 4MHz clock signals (even the one promoted as having a 6MHz CPU, honest!) so we'll call them **slow learners.** They do not come with any significant amount of demo or utility software. And they communicate with the host computer via RS 232, 9600 baud max. That's 1K byte/sec. Our board communicates over a parallel port with hardware AND software handshake, at 71K bytes/sec! We'll call those other boards **handicapped learners.**

Our board is definitely not for everyone. But some people find it very, very useful. Which group do you fit into?

DIGITAL ACOUSTICS
1415 E. McFadden, Ste. F
Santa Ana, CA 92705
(714) 835-4884

Apple, Applesoft and Apple II are trademarks of Apple Computer Company. Pet is a trademark of Commodore Business Machines.

FORTH for the 6809

by Ronald W. Anderson

The various versions of FORTH available for 6809-based computers are discussed. These include CCFORTH for the Color Computer, fig-FORTH, and several FLEX-based systems.

fig-FORTH

No discussion of FORTH would be complete without mention of the FORTH Interest Group (fig). Fig has implemented FORTH for most of the processors that are commonly used including the 6800, 6809, 6502, and 68000. Source listings of fig-FORTH for these processors are available from fig for a nominal charge (it was around $15 when I bought it).

The 6809 version is compatible at the most minimal level with the FLEX operating system. It is implemented as (or includes) its own disk operating system, accessing disk sectors directly. Depending on the number of bytes per sector, it utilizes four or eight sectors to form a unit of memory called a screen. A screen is a terminal screen full of information, consisting of 16 lines of 64 characters. The FORTH disk operating system is workable, but primitive by present day standards. There is no directory facility. Standard practice is to use the first line of every screen as a description of the contents, a type of comment line. A FORTH word will list the first lines of all the screens, forming a directory of sorts. What serves to make the use of this disk operating system a bit more difficult, is that the sectors written by fig-FORTH may not be accessed by or through the FLEX operating system (except through a sector dump utility).

Included in the fig-FORTH system is a line editor that allows you to access and change information stored in screens. Some of the more advanced systems have a screen editor that is a bit more convenient than the fig line editor.

When you write a program or application, you edit some screens, putting your word definitions on them. You may at any time load a range of screens, which causes them to be compiled so the program may be run. Therefore, you can debug your program interactively, running and editing screens alternately. There is a screen buffer that can hold from two to several screens, so they don't have to be written to or read from disks for every change. FORTH keeps track of whether a screen has been updated, and won't let it be overwritten in the screen buffer without automatically rewriting it to the disk. That is a very handy feature.

Fig-FORTH as supplied has the source code for FORTH and some FORTH screens that must somehow be bootstrapped into the system to get the editor working. The disk operating system ties to FLEX and is compatible at the disk driver level. I have the fig implementation and have typed it in and gotten it up and running. If you are new to FORTH, however, I would not recommend going the fig route. However, if you can learn FORTH first on another system, the fig route is an inexpensive, though time consuming, way to go.

FORTH from Talbot Microsystems

Ray Talbot, who wrote the fig implementation for the 6809, sells an implementation of FORTH through his company, Talbot Microsystems. Talbot's implementation, known as tFORTH, offers features that make it easier to use with FLEX. The disk containing the FORTH compiler has several tracks that are initialized in the standard FLEX format. This allows the user to have the FORTH compiler on those tracks and to call it with a standard FLEX call. The user may put FLEX itself, the necessary utilities to boot FLEX, and perhaps some utilities such as COPY, LIST, etc., on those tracks also. The remainder of the disk is used by FORTH directly as screen storage.

FORTH will access a second disk drive when the screen number requested goes beyond the range of those numbers available on the first drive (a handy feature). There are some utilities included in FORTH to do an index, list screens over a range of specified numbers and list them formatted three to a page for a printer. Other utilities are included to copy one screen to another and delete screens.

In addition to tFORTH, Talbot has available an extended version called tFORTH+. The added features are a screen-oriented editor, a full 6809 assembler, a CASE statement, and some additional data types such as ARRAYS. Another program available from Talbot is firmFORTH. firmFORTH allows you to shrink a finished application program to a bare minimum by including only the core words that you have used in your application. It eliminates everything else, and generates only the necessary binary code that you may burn into an EPROM for a dedicated application, or save on a disk for quick load and run.

XFORTH from Frank Hogg Labs

Frank Hogg Laboratories of Syracuse, NY, also supplies a FLEX-compatible version of FORTH called XFORTH. XFORTH includes a comprehensive manual that contains a tutorial on FORTH as well as an extensive FORTH Glossary. This implementation comes complete with many FORTH screens of application programs that aid you in using FORTH, including a complete double- and triple-precision integer arithmetic package, an extensive Screen Editor, and a complete 6809 assembler. Versions are provided for several terminal models. With a bit of study, they may be adapted to most any terminal that

allows computer-controlled cursor positioning, cursor home, and screen erase.

XFORTH varies from the fig standard disk operating system. It still uses the screen concept, but an application or program is kept in a standard FLEX file. To load a particular application, you load the FLEX file into a screen buffer. Now screens in the program may be accessed just as in standard FORTH. When desired, the program may be saved to a disk file. The advantage of this file-handling method is that the disk is completely compatible with FLEX utilities to get a DIRECTORY, LIST a file, COPY a disk, etc. It makes life a bit easier for someone who uses a number of different languages and wants to keep things on a more common base.

CCFORTH from Frank Hogg Labs

Frank Hogg Laboratories also offers a version of FORTH that runs on the TRS-80 Color Computer. It uses the TRS-80 disk operating system, so it is not necessary to have FLEX on the CC in order to use it. CCFORTH contains a screen editor and some other features that make it particularly well adapted to use with the Color Computer. The screen editor uses the four directional arrow keys on the CC to position the cursor in the command mode. You may also delete characters or lines and insert blank spaces or lines in this mode. By typing ENTER, you enter the insert mode; then you may type in lines of text or insert characters in blank spaces.

I used this editor for entering and testing the HYPOTENUSE program described in the companion article (p.). The three word definitions fit nicely on one screen. To get into the editor mode, type < screen number > EDIT. When you are done, type < screen number > LOAD, and your newly defined words are added to the dictionary. If you get any error messages, you can FORGET SQUARE (the first new word) and re-edit your definitions. This process is repeated until your program runs. Because of the display width limitation on the CC, CCFORTH screens are 32 lines of 32 characters rather than the standard 16 lines of 64 characters. I found I liked that arrangement better. The shorter lines allow for better program phrasing.

Charles Moore, the inventor of FORTH, has said that FORTH is about as controversial as religion or politics. He maintains that a good way to start an argument among programmers is to say "Boy, FORTH is a great language." I have mixed feelings about FORTH; it is difficult to learn, and reading someone else's program can be a problem (though I am beginning to see that FORTH programs can be readably written. Keeping track of all the data floating around on the stack can be most frustrating, particularly to a beginner. However, in spite of a few misgivings, I never cease to be impressed with how much you can do with FORTH with so little source code! My guess is that with equally well-documented and formatted programs in FORTH, Pascal, and BASIC, the FORTH program will be considerably shorter and will run considerably faster than any of the others.

68'FORTH for 6809 available as a printed listing from:

FORTH Interest Group
P.O. Box 1105
San Carlos, CA 94070

Write for their current catalog and prices.

tFORTH, tFORTH+, and firmFORTH available from:

Talbot Microsystems
1927 Curtis Ave.
Redondo Beach, CA 90278

tFORTH $100.00
tFORTH+ $250.00
firmFORTH $350.00
(requires but does not include tFORTH+)

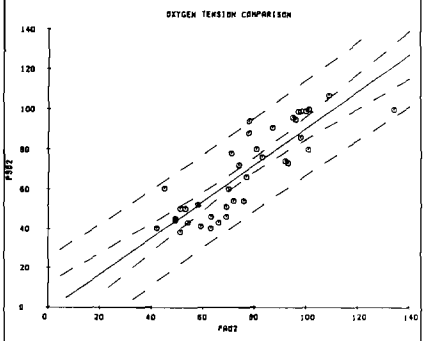
XFORTH, CCFORTH available from:

Frank Hogg Laboratories
130 Midtown Plaza
Syracuse, NY 13210

XFORTH $149.95
plus $2.50 shipping (huge manual)
CCFORTH ..... $ 99.95

The author may be contacted at 3540 Sturbridge Ct., Ann Arbor, MI 48105.

BIOSTATISTICS III



WATANABE PLOTTER GRAPHICS

BIOSTATISTICS III is a combined statistical analysis and graphical data plotting software package. It covers a comprehensive range of bivariate and multivariate tests commonly used in enigineering and the social and medical sciences.

SPECIAL FEATURES:
1) Generates regression plots in screen high resolution graphics and on the Wantanabe WX 4671 plotter.

2) Stores data in standard DOS 3.3 text files, easy to access from your own software or data base.

3) Programs are written in Applesoft Basic. Can be user modified.

4) Includes extensive data editing and file manipulation facilities.

5) Tabulated data printouts and statistical results.

STATISTICAL TESTS:	GRAPHICS
STANDARD DEVIATION	
PAIRED STUDENT (t) TEST	
UNPAIRED STUDENT (t) TEST	
WILCOXON PAIRED TEST	
MANN-WHITNEY U TEST	
A N O V A (ONE AND TWO WAY)	
LINEAR REGRESSION	*
EXPONENTIAL REGRESSION	*
POLYNOMIAL REGRESSION	*
CURVILINEAR REGRESSION	*
MULTIPLE REGRESSION	

SYSTEM REQUIREMENTS:
Apple 2 plus with 48k RAM and two Disk Drives. Also compatible with the Apple 3 in emulation mode.

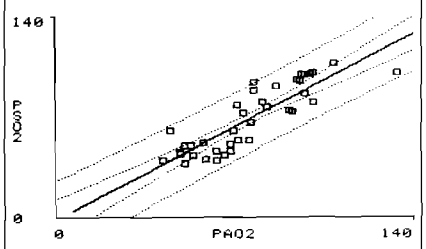
The package includes documentation and both program and data discs.

P R I C E : $95.00

Personal checks, COD or P.O.# on offical order forms, accepted. California residents add 6% sales tax.

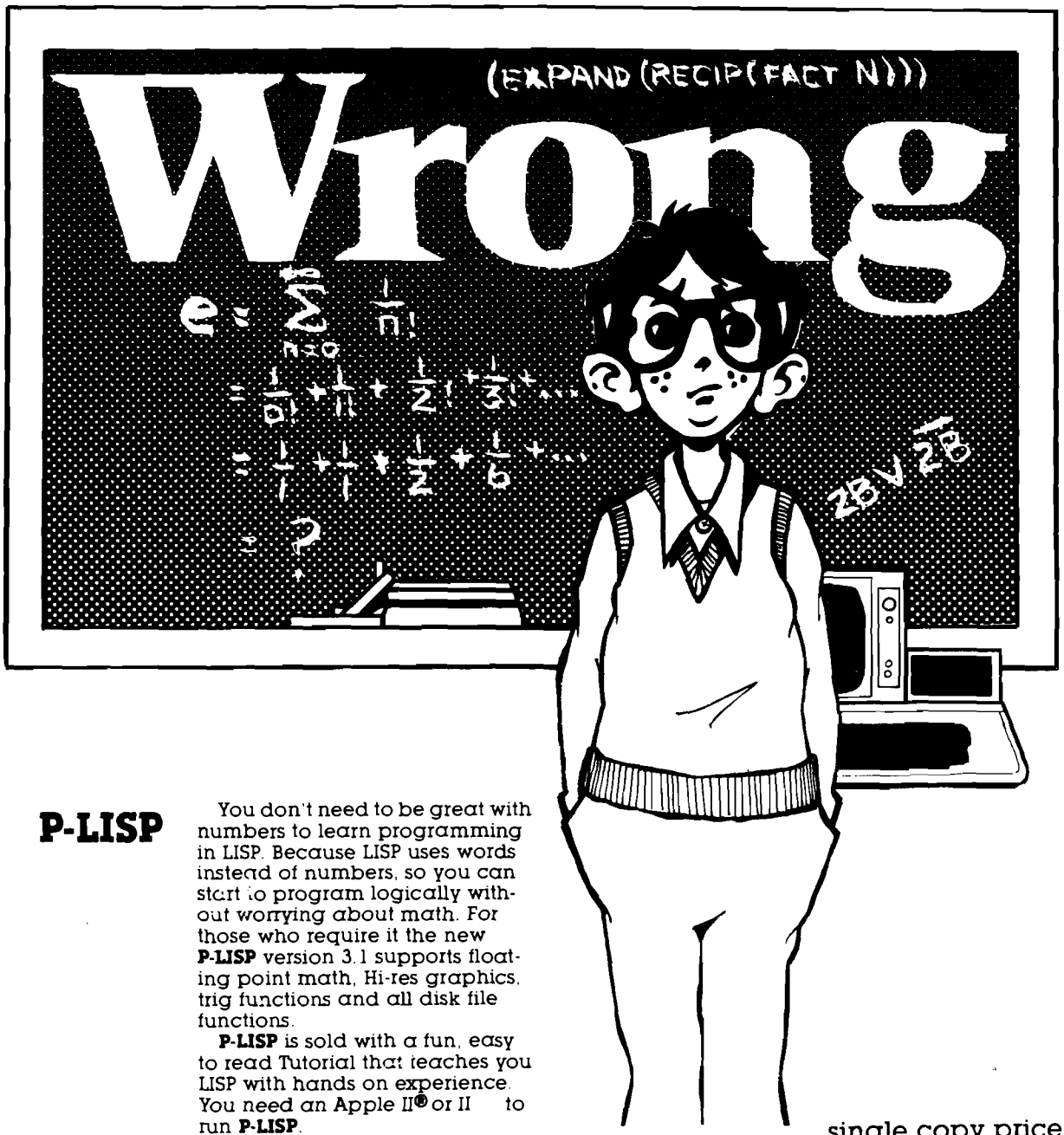
Write for more information or order from:

A2DEVICES P.O.BOX 2226 ALAMEDA CALIFORNIA 94501. TEL (415) 527-7380.



SCREEN GRAPHICS DUMP TO EPSON MX80

APPLE is a registered trademark of Apple Computers, Inc.
WATANABE is a registered trademark of Watanabe Corporation.
EPSON is a registered trademark of Epson America, Inc.



P-LISP

You don't need to be great with numbers to learn programming in LISP. Because LISP uses words instead of numbers, so you can start to program logically without worrying about math. For those who require it the new **P-LISP** version 3.1 supports floating point math, Hi-res graphics, trig functions and all disk file functions.

P-LISP is sold with a fun, easy to read Tutorial that teaches you LISP with hands on experience. You need an Apple II® or II to run **P-LISP**.

Educators: LISP is a great teaching language.
Write to us for special quantity pricing.

single copy price
$149.95
(includes tutorial)

gnosis

4005 Chestnut Street Philadelphia, PA 19104 215-387-1500
Toll Free (orders only) 800-523-0725

Apple II II is a registered trademark of Apple Computers Inc

The World According to LISP

by Steven Cherry

LISP has always been considered a language for large computers. In this introduction, learn about LISP as it is implemented on a microcomputer.

Many people who work with computers believe LISP is a strange and obscure language used only by mad computer scientists who are locked away in research laboratories or universities working in that nebulous field commonly known as Artificial Intelligence (AI). Indeed, because LISP generally has been restricted to use only on large computer systems, few have been able to obtain exposure to this language outside an AI or academic environment. However, with the introduction of P-LISP for the Apple II, as well as other LISP interpreters for other machines, this trend is rapidly changing. For Apple owners, access to a comprehensive and quite powerful LISP interpreter is now at their fingertips. It is therefore a worthwhile endeavor to take a close look at LISP to see what we've been missing for so long.

LISP stands for "LISt Processor." It was developed by John McCarthy at MIT in the late 1950's, originally as a tool for mathematical research. Because of its unique features, LISP quickly caught on as the language of choice for work in Artificial Intelligence. Unlike Pascal or FORTRAN, there is no real LISP standard. Today LISP is available in various flavors, such as INTERLISP, MACLISP, MTSLISP, and, of course, P-LISP. However, all LISPs more or less look and act the same and are descendants of McCarthy's LISP 1.5.

What makes this language so unique? The major strengths and features of LISP can be outlined as follows:

- LISP uses the same data structure to represent both programs (actually functions) and data. Since programs and data are indistinguishable (as far as the LISP interpreter is concerned), it is relatively easy to write LISP programs that construct and execute other LISP programs.
- LISP is interpreted, providing the user with immediate response. The interactive nature of LISP makes it easier to develop and debug programs and gives the user the feeling that he is talking to the computer.
- LISP has a simple and uniform syntax; there are only a few rules to remember, and these quickly become second nature. Moreover, LISP is independent of the details of the machine on which it is running; there is no need to deal with word sizes, the lengths of variable names, declarations, etc.
- LISP is ideal for applications requiring symbolic manipulation because, unlike BASIC or Pascal, LISP deals with objects rather than strings, records, variables, pointers, etc. All of the bookkeeping necessary for representing or manipulating these objects is implicit in the language; for example, a program to differentiate polynomials might be a chore to write in BASIC or Pascal, but is relatively simple and straightforward in LISP.
- LISP is a heirarchical language; LISP programs are actually functions, each constructed out of more primitive functions. A LISP interpreter is little more than a set of pre-defined functions. It is thus possible to build entire systems (for example, the programming language Smalltalk) out of LISP primitives, and other systems on top of these, etc.

So the picture doesn't appear too perfect, I must point out that, as with any language, LISP has some drawbacks as well. Chief among these is its memory demands: this language is a memory hog and, although it can fit on a micro (P-LISP is roughly 14K in length and supports about 80 functions), this demand can be quite limiting, depending on the specific application involved. Performance is another factor; because LISP is interpreted, execution speed is much slower than that of a compiled language such as Pascal (however, some systems do have LISP compilers available). A third problem is the simple syntax of LISP, which can be a liability as well as an asset. It is quite easy to write a LISP function that is so hopelessly obscure as to baffle even the person who wrote it.

The Structure of LISP

The basic unit of information in LISP is the *atom*. There are *literal atoms*, which are represented as a sequence of alphanumeric characters beginning with a letter, and *numeric atoms*, which are simply numbers. Thus, A, HELLO, PQR57, and WALRUS are literal atoms, while 12 and −56.87 are numeric atoms. 15AYT would not be an atom. The sequence of characters that denote an atom are called the atom's *print name*. For example, the print name of atom DEF is the sequence of characters D, E, and F.

Atoms can be combined to form the basic data structure of LISP, the *list*. A list is simply a sequence of *symbolic expressions* or *s-exprs*, bound by a pair of parentheses, where a s-expr is defined to be an atom or a list. So, (A B C) is a list comprised of three s-exprs, the atoms A, B, and C. Similarly, (HAIL

AND (WELL MET)) is a list comprised of three s-exprs — namely, the atom HAIL, the atom AND, and the list (WELL MET), which in turn is comprised of the two atoms WELL and MET. A list may contain any number of atoms or lists as its elements. Be aware that the parentheses are not part of the list; they are punctuation marks that define the list (just as a pair of quotes defines a string in BASIC).

At the heart of LISP is the *evaluator.* Whenever you type something into LISP, the interpreter tries to evaluate what you typed in and return the result (this is known as a READ-EVAL-PRINT loop). If, for some reason, LISP cannot evaluate your input, it will give you a friendly error message indicating where it got stuck and why. Actually, error recovery in LISP is entirely implementation-dependent; there is nothing in the definition of the language that specifically states what course of action should be taken if an error condition arises.

The following rule is used for evaluating lists: when you give LISP a list to evalute, LISP treats the first element of the list as the name of a function, and the remaining elements (if any) as the arguments to the function. So, if you type the list (A B C), LISP will try to apply some function named A to the arguments B and C. For example, suppose you want to add two numbers together. In BASIC, you would type something like PRINT 1 + 2 to add 1 and 2. In LISP, you can accomplish this by typing (ADD 1 2). ADD is a built-in function that takes two arguments — namely, two numeric atoms — and returns the sum of the arguments (in this case 3). The value of a s-expr is the value returned when the s-expr is evaluated. So the value of (ADD 1 2) is the atom 3.

What if you type in something that doesn't make much sense, like (ADD 1)? Since the function ADD expects two arguments, you should get an error message. P-LISP will give the following:

```
** ERROR: TOO FEW ARGS **
    ADD:: (1)
```

The first line of the message indicates what's wrong, and the next line shows the function and the list of arguments LISP was working on when the error occurred.

Atoms may have values associated with them, just like a variable in BASIC or Pascal can be assigned a value. If you type an atom into LISP, the interpreter will return the value of the atom, if it has one. For example, numeric atoms are defined to have themselves as their value. The value of the atom 3 is 3 (as you'd expect). If you type 3 into LISP, you'll get back 3, the same as if you'd typed (ADD 1 2). As a general rule, literal atoms don't have a value until they're given one. The value cf a literal atom may be any s-expr.

One thing you can do with lists is take them apart. The LISP function CAR takes a list and returns the first element of the list. If you type (CAR '(A B C)) you get A as the value. When you give LISP something to evaluate, of the form (Function Arg1 Arg2 . . . Argn) LISP first evaluates the arguments, *then* applies the function to the argument values to return the final result. For example, if the value of A is 1, and the value of B is 2, then (ADD A B) will return 3.

In the example for CAR above, the quote tells LISP *not* to evaluate the argument before applying the CAR function. In other words, if you didn't type the quote, LISP would first try to evaluate the s-expr (A B C), then apply CAR to the result. With the quote, LISP applies CAR directly to (A B C), returning A.

The LISP function CDR is the complement of CAR. CDR takes a list and returns that list minus the first element. So, (CDR '(A B C)) returns (B C). If you type (CDR '(HITHERE)) LISP gives NIL. NIL is a list containing zero elements, or the empty list. Also, NIL is represented by an empty pair of parentheses, ( ). NIL is considered a special element of LISP because it is both a list and an atom. NIL has itself as its value; i.e., the value of NIL is NIL.

NIL also is used to represent the truth value "false". As you'd expect, there is an atom to represent the truth value "true" — namely, T. The value of T is T. NIL and T are the only literal atoms built into LISP with predefined values (actually, any non-NIL value in LISP is considered to represent "true". T is convenient because its value is always guaranteed to be non-NIL).

One area where T and NIL come into play is with *predicates.* Predicates are functions that perform a certain test on their arguments and return T if the argument passes the test and NIL if it fails. One such predicate is ATOM; ATOM returns T if its argument is an atom, and NIL otherwise. For example, (ATOM '(IM A LIST)) returns NIL, whereas (ATOM 'BOMB) returns T. The s-expr (ATOM ( )) also returns T (remember, ( ) is an alternate representation for NIL, which is an atom).

Another important predicate is NULL, which returns T if its argument is NIL, and NIL otherwise. So, (NULL '(8 9 10)) returns NIL, but (NULL (CDR '(BLEAT))) returns T. LISP first evaluates the arguments to a function, then applies the function to the result. In the above example, the argument to NULL is the s-expr (CDR '(BLEAT)). LISP evaluates this and returns NIL, which is then passed on to NULL. The NULL of NIL is T, so T is returned as the value of the entire s-expr.

Incidentally, CAR and CDR are the only functions in LISP whose names have nothing to do with their meaning. Their names are derived from the hardware on which the first LISP interpreter was implemented: CAR stands for "Contents of Address Register" and CDR stands for "Contents of Decrement Register." On some LISP systems they have been given the more meaningful names FIRST and REST; but for the most part, their original names have stuck.

In addition to functions that take lists apart, LISP is supplied with functions to put lists together. One of these is the function CONS, for CONStruct. CONS takes two s-exprs and returns a new list such that the first argument is the CAR of the list and the second argument is the CDR of the list. For example, (CONS 'THIS '(IS FUN)) returns (THIS IS FUN). Note that the CAR of this list is THIS, and the CDR is the list (IS FUN).

CONS puts things at the front of lists. Another function, APPEND, puts things at the end of lists. For example, (APPEND '(THESE THAT) 'THOSE) returns (THESE THAT THOSE).

Other functions used for building lists include CONC and LIST. CONC concatenates lists together and LIST creates a list of its arguments. Some examples: (CONC '(I LISP) '(YOU LISP)) returns (I LISP YOU LISP), and (LIST 'A 'B '(AND C)) returns (A B (AND C)).

I mentioned above that it is possible to assign values to literal atoms. This is done with the LISP function SETQ. The first argument to SETQ is the atom to be assigned a value and the second argument is the value. An anomaly to remember about SETQ is that the first argument is not evaluated, but the second argument is. For example, if you want to assign the atom A the value

(WHO AM I), type (SETQ A '(WHO AM I)). Now whenever the atom A is evaluated, the list (WHO AM I) would be returned.

Although there is no string data type in LISP, there are facilities to treat atoms' print names as though they were strings. One such facility is the special kind of literal atom called the *string atom*. A string atom is the same as a literal atom in all respects, except that its print name is delimited by a pair of double quotes. Any character may appear within the quotes (except a double quote). For example, "THIS IS A SINGLE ATOM" is a single atom whose print name is the sequence of characters between the double quotes. String atoms allow you to create atoms with "funny" print names; for instance, "(" can be an atom (it has nothing to do with the (symbol used to denote lists).

The LISP functions EXPLODE and IMPLODE allow the user to manipulate print names. EXPLODE takes an atom and returns a list of the characters in the atom's print name; thus, (EXPLODE 'TNT) returns the list (T N T). IMPLODE does the reverse of EXPLODE; it takes a list of atoms and returns an atom whose print name is the compression of the atoms in the list. For example, (IMPLODE '(N E U T R O N)) returns the atom NEUTRON.

Of course, LISP wouldn't be much fun or very useful unless you were able to create your own functions. The LISP function DEFINE is used to define functions. The general form of a function definition is as follows:

```
(DEFINE (function-name (LAMBDA
(formal arguments)
        function-body
    )))
```

The *function-name* is a literal atom, which is what the function will be called. A *LAMBDA-expression* follows the function-name; all user-defined functions must be some form of LAMBDA-expression. The second part of a LAMBDA-expression (following the atom LAMBDA) is a list of atoms that are the function's *formal arguments* (if the function takes no arguments, the list is NIL). This list tells LISP the number of actual arguments the function takes, and how these arguments are referred to in the body of the function (the example below will help clarify these points). The last part of the LAMBDA-expression is the actual body of the function and is a s-expr whose value will be returned as the value of the function.

As an example, suppose you want a function that returns the second element of a list; that is, if you gave the function the list (A B C), you want the function to return B. Call this function SECOND:

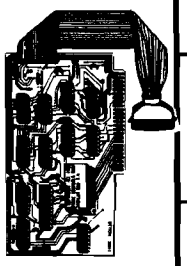
```
(DEFINE (SECOND (LAMBDA (L)
        (CAR (CDR L) )
    )))
```

L is the formal argument of SECOND. When SECOND is invoked, L is *bound* (SETQed) to the value of the actual argument and the function body (the CAR of the CDR of L) is evaluated. For example, if you type (SECOND '(A B C)), L is bound to the list (A B C), and CAR (CDR L)) is evaluated, returning B.

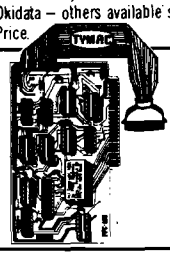
Whenever a user-defined function (a LAMBDA-expression) is invoked, a *local environment* for the function is created consisting of the formal arguments of the function and the values they are bound to (known as *LAMBDA-binding*) at the time of the invocation.

APPLE HARDWARE

THE TACKLER ™ – DUAL • MODE PARALLEL INTERFACE FOR THE APPLE® 2 BOARDS IN ONE FOR NO MORE COMPATIBILITY PROBLEMS!



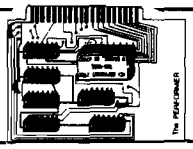
An intelligent board to provide easy control of your printer's full potential. Plus a standard parallel board at the flip of a switch – your assurance of compatibility with essentially all software for the APPLE®. Hires printing with simple keyboard commands that replace hard to use software routines. No disks to load. Special features include inverse, doubled, and rotated graphics and many text control features, available through easy keyboard or software commands. Uses Industry standard graphics commands. This is the first truly universal intelligent parallel interface! Change printers – no need to buy another board. Just plug in one of our ROM'S and you're all set. ROM'S available for Epson, C. Itoh, NEC, and Okidata – others available soon. Specify printer when ordering. Call for Price.



THE UPGRADEABLE PPC-100 PARALLEL PRINTER CARD

A Universal Centronics type parallel printer board complete with cable and connector. This unique board allows you to turn on and off the high bit so that you can access additional features in many printers. Easily upgradeable to a fully intelligent printer board with graphics and text dumps. Use with EPSON, C. ITOH, ANADEX, STAR-WRITER, NEC, OKI and others with standard Centronics configuration. **$139.00**

IF YOU WANT GRAPHICS AND FORMATTING THEN CHOOSE THE PERFORMER



for Epson, OKI, NEC 8023, C. ITOH 8510 provides resident HIRES screen dump and print formatting in firmware. Plugs into Apple slot and easy access to all printer fonts through menu with PR# command. Use with standard printer cards to add intelligence. **$49.00** specify printer.

THE MIRROR FIRMWARE FOR NOVATION APPLE CAT II®

The Data Communication Handler ROM Emulates syntax of an other popular Apple Modem product with improvements. Plugs directly on Apple CAT II Board. Supports Videx and Smarterm 80 column cards, touch tone and rotary dial, remote terminal, voice toggle, easy printer access and much more. List $39.00 Introductory Price **$29.00**

MINI ROM BOARDS

Place your 2K program on our Mini Rom Board. Room for one 2716 EPROM. Use in any slot but zero. Only **$34.95**

DOUBLE DOS Plus

A piggy-back board that plugs into the disk-controller card so that you can switch select between DOS 3.2 and DOS 3.3 DOUBLE DOS Plus requires APPLE DOS ROMS. **$39.00**

APPLE SOFTWARE

Super Pix

Hires screendump software for the Epson, OKI, C. Itoh and Nec 8023. Use with Tymac PPC-100. Special **$19.95** (Specify Printer)

***Mr. Lister* – Customer Contact Profiler & Mailer**

A Super Mail List Plus more — up to 1000 Entries on single 3.3 Disk (only 1 Drive required) — 2 second access time to any name — full sort capabilities — Dual Index Modes — supports new 9 digit Zip. Easy to follow manual — Not Copy Protected — 4 user defined tables with 26 sort selections per table — Beta tested for 6 months — user defined label generation. Introductory Price $135. **$99.00** Dealer &: Dist. Inquires Invited.

APPLE LINK

A communications system for the Apple® (Requires Hayes Micro Modem). Transmit and receive any type of file between APPLES®. Automatic multi-file transfer, real time clock indicating file transfer time. Complete error check. Plus conversation mode. Only one package needed for full transfers. Compatable with all DOS file types. (requires Hayes Micro Modem) **$59.00**

THE APPLE CARD/ATARI CARD

Two sided 100% plastic reference card Loaded with information of interest to all Apple and Atari owners. **$3.98**

NIBBLES AWAY II

AGAIN! Ahead of all others.

- **AUTO-LOAD PARAMETERS . . .** Free's the user from having to Manually Key in Param values used with the more popular software packages available for the Apple II.
- **EXPANDED USER MANUAL . . .** incorporates new Tutorials for all levels of expertice; Beginners Flowchart for 'where do I begin' to 'Advanced Disk Analysis' is included.
- **TRACK/SECTOR EDITOR . . .** An all new Track/Sector Editor, including the following features: Read, Write, Insert, Delete Search, and impressive Print capabilities!
- **DISK DIAGNOSTICS . .** Checks such things as: Drive Speed, Diskette Media Reliability, and Erasing Diskettes.
- **HIGHEST RATED . . .** Best back up Program in Softalk Poll (Rated 8.25 out of 10).
- **CONTINUAL UPDATES . . .** Available from Computer Applications and new listings on the source. **$69.95**

Dealer and Distributor Inquiries Invited.

VISA MasterCard

MICRO-WARE DIST. INC.
P.O. BOX 113 POMPTON PLAINS, N.J. 07444
201-838-9027

This local environment remains in effect until the function is exited. At that time, the environment is destroyed and the next most recent environment b comes the "current" environment. The highest-level environment (the one in effect before any functions are invoked) is called the *global* environment.

Suppose you type (SETQ L '(THIS BETTER WORK)) into LISP. The global value of L is now set to the list (THIS BETTER WORK). If you type (SECOND '(WHOS ON FIRST)), a local environment for SECOND is created in which a *local* L is bound to the list (WHOS ON FIRST). The function body is then evaluated. When the function is exited, the local environment is destroyed and the value of the function body, the atom ON, is returned. If you now type L you get what you had before: (THIS BETTER WORK), which is the global value of L.

One of the nicer features of LISP is that it allows the definition of recursive functions. A recursive function is a function that is defined in terms of itself. A perfect example is the factorial function, which is defined as follows:

```
n! = 1 if n = 0
     n * (n - 1)! otherwise
```

This function can be represented very nicely in LISP:

```
(FACTORIAL (LAMBDA (N)
 (COND
  ( (EQUAL N 0) 1)
  (T (MULT N (FACTORIAL (SUB N 1) ) ) )
 )
)
```

COND is the LISP CONDitional construct and is analogous to the IF-THEN-ELSE construct in BASIC or Pascal. The form of the COND is as follows:

```
COND (e1 s1)
       (e2 s2)
        '' ''
       (en sn) )
```

which can be thought of as meaning

```
IF e1 THEN s1
ELSE IF e2 THEN s2
ELSE IF e3 THEN s3
 ''   '' ''   ''   ''
ELSE IF en THEN sn
ELSE NIL
```

Each e1 is evaluated until one evaluates to a non-NIL value. The corresponding s1 is then evaluated and returned as the value of the COND. If all of the ei evaluate to NIL, then COND returns NIL.

In the FACTORIAL function above, the COND returns 1 if N is equal to 0. Otherwise, the MULT s-expr is evaluated (note that the T forces this s-expr to be evaluated if the first test fails). The MULT contains a recursive call on FACTORIAL; the argument passed to FACTORIAL here is N - 1. The value returned by this call is multiplied by the "current" value of N (remember that each time FACTORIAL is invoked, a new "local" N is created and bound to the actual argument).

A handy way to observe the evaluation of a function is *via* a *function trace*. Most decent LISPs are supplied with some mechanism for tracing functions, an indispensable debugging aid. A trace of a function displays the arguments passed to the function when it is invoked and the value returned by the function when it is exited. For example, in P-LISP, if you are tracing the ADD function and type (ADD 4 5), you would see

```
---> >   ADD:: (4 5)
< <---   ADD:: 9
```

The ---> > arrow indicates a function entering and the < <--- arrow indicates a function exiting.

Suppose you want to see how the evaluation of FACTORIAL proceeds. If you trace the function and type (FACTORIAL 4), you will see the following:

```
--->> FACTORIAL:: (4)
  (level 1 — value of N is 4)
--->> FACTORIAL:: (3)
  (level 2 — value of N is 3)
---> > FACTORIAL:: (2)
  (level 3 — value of N is 2)
---> > FACTORIAL:: (1)
  (level 4 — value of N is 1)
---> > FACTORIAL:: (0)
  (level 5 — value of N is 0)
< <--- FACTORIAL:: 1
  (level 5 — 1 is returned)
< <--- FACTORIAL:: 1
  (level 4 — 1 is returned)
< <--- FACTORIAL:: 2
  (level 3 — 2 is returned)
< <--- FACTORIAL:: 6
  (level 2 — 6 is returned)
< <--- FACTORIAL:: 24
  (level 1 — 24 is returned)
```

Note that each level has its own "local" N, and the value of this N is multiplied by the value returned by FACTORIAL from the next level down.

Although recursion is usually a simple and elegant way to solve a programming problem, it is not always the best method to use. In particular, recursion tends to be inefficient. A great deal of overhead is involved when entering a function and setting up the local environment, both in execution speed and memory consumption. If these factors are critical, such overhead should be kept to a minimum. Luckily, LISP is provided with an iterative programming capability called PROG. Here is the definition of FACTORIAL using a PROG instead of recursion:

```
(FACTORIAL (LAMBDA (N)
 (PROG (PROD)
  (SETQ PROD 1)
  LOOP
   (COND
    ( (EQUAL N 0) (RETURN PROD) )
   )
   (SETQ PROD (MULT PROD N) )
   (SETQ N (SUB N 1) )
   (GO LOOP)
 )
)
```

The first part of PROG is a list of local atoms to the PROG. In the example above, PROD is declared a local atom. Local atoms exist only in the context of the PROG and disappear when the PROG is exited. When the PROG is entered, their values are intially SETQed to NIL.

Each s-expr in a PROG is evaluated in succession unless the s-expr is an atom. Such atoms are considered *labels* and are not evaluated (i.e., they are skipped). In the previous example, the atom LOOP is a label. The GO function causes flow-of-control to proceed to the label indicated in the GO (not unlike a GOTO in BASIC).

The RETURN function is used to exit a PROG. The argument of the RETURN is evaluated and returned as the value of the PROG. If RETURN is not used to exit the PROG, then PROG returns NIL.

The example given for FACTORIAL should now be easy to understand. When the PROG is entered, PROD is SETQed to 1 and the main loop is then entered. The first part of the loop is a test. If N is 0, the PROG is exited and PROD is returned; otherwise PROD is multiplied by N and N is decremented

(SUB is the subtract function). GO then brings you back to the beginning of the loop. As you can see, the iterative FACTORIAL is functionally equivalent to the recursive version. The major difference is that although the iterative function is longer, it is probably faster and more efficient than the recursive one.

LISP functions that are built into the interpreter are called *SUBRs*, for SUBRoutine. Functions that are defined by the user are called *EXPRs*. The EXPRs that you have examined so far are functions that take a known number of arguments, as indicated by the function's formal argument list.

The LISP mechanism for defining a function for which the number of arguments is not known is the *FEXPR*. FEXPRs are defined using the following format:

```
(DEFINE (function-name (FLAMBDA
(formal-argument)
            function-body
      )))
```

The difference between a FEXPR and an EXPR is that a FEXPR contains a *single* formal argument in the formal argument list. When such a function is invoked, the formal argument is bound to the entire unevaluated list of actual arguments. For example, define the following function:

```
(DEFINE (PRINTME (FLAMBDA (X)
      X)))
```

If you now type (PRINTME LISP LISPER LISPEST), you get (LISP LISPER LISPEST). The formal argument X gets bound to the list of actual arguments (LISP LISPER LISPEST). Note these arguments are *not* evaluated. The function then just returns X.

If you want to define a function that adds up an arbitrary number of numeric atoms, the following functions will serve your needs:

```
(DEFINE (ADDLIST (FLAMBDA (X)
      (ADDLIST2 X))))

(DEFINE (ADDLIST2 (LAMBDA (X)
 (COND
  ((NULL X) 0)
  (T (ADD (CAR X) (ADDLIST2 (CDR X)))
 ))))
```

Since the function can take any number of arguments, it has to be declared a FEXPR. This function, ADDLIST, simply passes the argument list to the function ADDLIST2, which recursively adds up the elements of the list. Note that a FEXPR should *never* be recursive, since its arguments are never evaluated.

The final LISP feature discussed in this article is the *property list*, or *p-list*. A p-list is a list of properties and property values that may be associated with a literal atom. A p-list has the form

(prop1 value1 prop2 value2 ... propn valuen)

For example, you may want to assign the property COLOR to the atom BALL with the property value RED. The LISP function PUT is used to put properties and values on an atom's p-list. Properties must be literal atoms; a property value may be any s-expr. The s-expr (PUT 'BALL 'COLOR 'RED) puts the property COLOR with value RED on the p-list for BALL.

The LISP function GET is used to retrieve property values. If you type (GET 'BALL 'COLOR), you will get RED as the result. GET returns NIL if the atom does not have the indicated property on its property list.

Earlier I mentioned that the LISP function CDR took a list as its argument. CDR also can take an atom. The CDR of a literal atom is its property list. If you type (CDR 'BALL), you'll get (COLOR RED). The function REM removes properties from a p-list. Thus (REM 'BALL 'COLOR) removes the property COLOR from BALL's p-list.

What are p-lists good for? Suppose you want to create a dictionary of English words to be used by a natural language processor. The various properties associated with a word, such as part of speech, plural form, etc., could be conveniently stored on a p-list. For example, the p-list for the atom APPLE could look like (NOUN T PLURAL APPLES OBJECT FRUIT SHAPE ROUND). These properties could be used by the natural language parser to determine if a statement was syntactically correct and semantically meaningful.

P-lists also are used to store function definitions. LISP determines whether or not an atom is a function by checking the atom's p-list for the EXPR or SUBR property. If the function is an EXPR, the property value is the function definition; if it's a SUBR, the property value is implementation-dependent (usually the address of the interpreter subroutine that evaluates the SUBR). One way to have a LISP function create other functions on the fly is by PUTting the function definitions on atom p-lists.

There is still a great deal about LISP and its applications that are beyond the scope of an introductory article. Hopefully, your curiosity is piqued enough to want to learn more about this unique and fascinating programming language. The P-LISP package, published by Gnosis, Inc., is a worthwhile investment for those who own an Apple and want to become part of the world of LISP.

References

1. Allen, John, *Anatomy of LISP*, McGraw-Hill, 1978.
2. Bagley, Steven and Shrager, Jeff, *The P-LISP Tutorial*, Gnosis, Inc., Philadelphia, PA, 1982.
3. Horn, B. and Winston, Patrick, *LISP*, Addison-Wesley, 1981.
4. Siklossy, L., *Let's Talk LISP*, Prentice-Hall, 1976.
5. Winston, Patrick, *Artificial Intelligence*, Addison-Wesley, 1977.

You may contact the author at 1041 Penn Circle F-606, King of Prussia, PA 19406.

MICRO™

C64 FORTH
for the
Commodore 64

Fig.-Forth implementation including:

- Full feature screen editor and assembler
- Forth 79 Standard Commands with extensions
- High resolution, 16 color character and sprite graphics
- Full I/O allowing IEEE cartridge and Basic data file compability
- Three voice tone and music synthesizer
- Detailed manual with examples and BASIC-FORTH conversions
- Trace feature for Debugging

$99.95 - Disk Version
(Works with 1540 or 1541 Disk)
or Cassette Version
(Commodore 64 is a trademark of Commodore)

PERFORMANCE MICRO PRODUCTS
770 Dedham Street, S-2
Canton, MA 02021
(617) 828-1209

MICRO™

New Publications

Programming the PET/CBM, by Raeto Collin West. Compute! Books (625 Fulton Street, P.O. Box 5406, Greensboro, NC 27403), 1982, 504 pages, paperback.
ISBN: 0-942386-04-3 $24.95

The 68000: Principles and Programming, by Leo J. Scanlon. Howard W. Sams & Co., Inc. (4300 W. 62nd St., Indianapolis, IN 46268), 1982, 238 pages, paperback.
ISBN: 0-672-21853 $14.95

Educational Software Directory: A Subject Guide to Microcomputer Software, compiled by Marilyn J. Chartrand and Constance D. Williams. Libraries Unlimited (P.O. Box 263, Littleton, CO 80160-0263), 1982, 292 pages, paperback.
ISBN: 0-87287-352-8 $22.50

Collegiate Microcomputer, Quarterly Journal starting February, 1983. Contact Brian J. Winkel, Editor, Rose-Hulman Institute of Technology, Terre Haute, IN 47803 for subscription information.

TRS-80 Color BASIC, by Bob Albrecht. John Wiley and Sons, Inc. (605 Third Ave., New York, NY 10158), 1982, 376 pages, paperback.
ISBN: 0-471-09644-X $9.95

A Structured Approach to Pascal, by Billy K. Walker. Richard D. Irwin, Inc. (Homewood, IL 60430), 1983, 209 pages, paperback.
ISBN: 0-256-02827-3 $9.50

Apple Files, by David Miller. Reston Publishing Co. (Reston, VA), 1982, 414 pages, hard cover.
ISBN: 0-8359-0192-0 $14.95

So You Are Thinking About a Small Business Computer, 1982/83 Edition, by R.C. Canning and N.C. Leeper. Prentice Hall, Inc. (Englewood Cliffs, NJ), 1982, 203 pages, paperback.
ISBN: 0-13-823617-8 $10.95

Teaching Computer Programming to Kids and Other Beginners, by Royal W. Van Horn. Sterling Swift Publishing Co. (1600 Fortview Rd., Austin, TX 78704), 1982, 142 pages, paperback.
ISBN: 0-88408-154-0 $9.95
plus $1.45 S/H

Software Blueprint and Examples, by Yaohan Chu. Lexington Books (D.C. Heath and Co., 125 Spring St., Lexington, MA 02173), 1982, 519 pages, hard cover.
ISBN: 0-699-05329-5 $39.95

MICRO™

PET / CBM™

SOFTWARE SELECT!

8032 DISPLAY OR **4032 DISPLAY**

FROM THE KEYBOARD OR PROGRAM
NOW RUN WORD PRO 3 OR WORD PRO 4

FROM THE SAME MACHINE

Available for either 4000 or 8000 Series

ALSO:
For **2001 / 3000** Series Computers

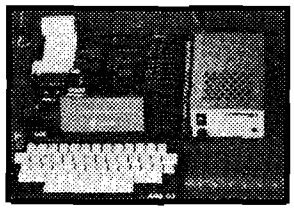
Operate these Models in a Full **8032** Like Display For Word Pro 4*
and all other 80 Column Software
All installation instructions included.

EXECOM CORP.
1901 Polaris Ave.
Racine, WI 53404
Ph. 414-632-1004

PET/CBM a trademark of Commodore Business Machines
*trademark of Professional Software, Inc.

AIM + POWER
from **COMPUTECH**

All prices **postpaid** (Continental U.S.- otherwise $2 credit)



Check the **outstanding documentation** supplied with AIM65!

Top quality power supply designed to Rockwell's specs for fully populated AIM65 — includes overvoltage protection, transient suppression, metal case and power cable:
PSSBC-A (5V 2A Reg; 24V .5A Avg, 2.5A Peak, Unreg) $64.95
Same but an extra AMP at 5 volts to drive your extra boards:
PSSBC-3 (5V 3A Reg; 24V .5A Avg, 2.5A Peak, Unreg) $74.95

The **professional's choice** in microcomputers:
AIM65/1K RAM ..... $429.95 BASIC (2 ROMS) $59.95
AIM65/4K RAM ..... $464.95 ASSEMBLER (1 ROM) $32.95
FORTH (2 ROMS) ..... $59.95.

SAVE EVEN MORE ON COMBINATIONS
AIM65/1K+PSSBC-A ... $479.95 AIM65/4K+PSSBC-3 ... $524.95
We gladly quote on all AIM65/40 and RM65 items as well.

ORDERS: (714) 369-1084
P.O. Box 20054 • Riverside, CA 92516
California residents add 6% sales tax

VISA master charge

MICRO™

PET Vet

By Loren Wright

Sound on the Commodore 64

My December column (MICRO 55:54) covered the exciting graphics features of the Commodore 64, including sprites and high-resolution graphics. The Commodore 64's sound includes many capabilities found only on dedicated synthesizers. In this column I explore the C64's sound features and then review some sound software.

The Problems

The sound capabilities of most computers, and even most add-on boards, are limited. Sounds programmed with these usually don't come close to their natural counterparts. It's not surprising, considering the way most computer sound is produced.

According to the theories of a physicist named Fourier, all sounds can be constructed by combining sine waves (figure 1) of the right frequencies. Full-fledged synthesizers allow full control over how much of each sine wave is used. In addition, the phase of each can be varied. As a result, synthesizers can duplicate nearly any sound.

How is music usually produced with home computers? On the PET and many others it is possible to set up a square wave on the CB2 line of a 6522 (VIA) by using an internal timer and the serial shift register. A square wave sounds a bit like a clarinet. The VIC and Atari computers use more sophisticated systems, with special chips partially dedicated to music.

Musical sounds are usually composed of a single, fundamental frequency that determines the pitch of the note, and various amounts of the harmonics, or multiples, of the fundamental frequency. For instance, a triangle wave consists only of odd harmonics, each in a proportion determined by the reciprocal of the square of its harmonic number. Therefore, a triangle wave is dominated by the fundamental, with the third harmonic only 1/9 as loud, the fifth only 1/25 as loud, and the other odd harmonics much softer. It is difficult to produce a sine wave with inexpensive digital circuitry. The more attainable triangle waveform looks and sounds similar to the sine wave, and although you can certainly hear the difference, it is a satisfactory replacement.

The Commodore 64 has three voices, each of which can be programmed with a triangle, rectangular, sawtooth, or random noise waveform, shown in figure 2. The rectangular wave's pulse width is programmable. The 6581 (or SID) chip has a great number of other capabilities, including filtration, synchronization, and ring modulation, which I'll discuss later.

The Envelope Please

Natural sounds seldom start immediately with their full volume, nor do they end abruptly. Think of the way they are produced. With a piano, a little hammer hits a string; with a violin, the hairs of the bow catch the string; with a wind instrument, the impact of the player's tongue starts the sound. How can a computer simulate the natural variations of volume within a single note?

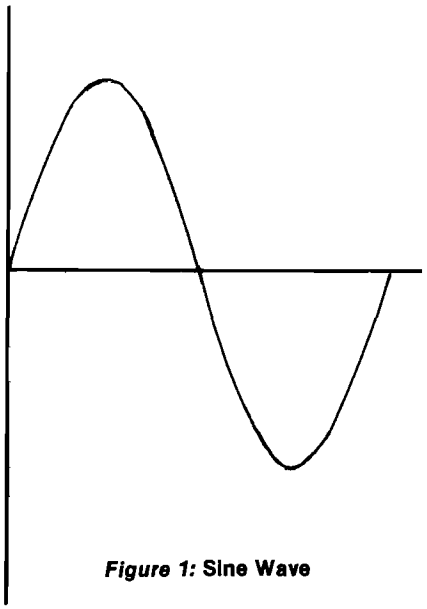


Figure 1: **Sine Wave**

Most computers don't. The Commodore 64 and real synthesizers use what is called an *envelope* for each note (see figure 3). There are four components of the envelope: *attack*, *decay*, *sustain*, and *release*. *Attack* is the time it takes for the note to increase from no volume to its maximum volume. *Decay* is the time it takes the volume to decrease to the *sustain* level, which is maintained for most of the duration of the note. Finally, *release* is the time it takes to go from the sustain volume level to silence again.

The shape of the envelope can be programmed. The durations of the attack, decay, and release portions can each be programmed to sixteen different values. The volume level of the sustain portion can be set to sixteen different proportions of the peak volume. There is a *gate* bit for each voice. When this bit is set to 1 the attack begins, followed by the decay and the sustain, where it stays until the gate bit is reset to 0. Then the release portion of the envelope begins.

This information is all you really need to know to use the 64 for simple music programming. Just select the frequency (this requires two POKEs), the waveform, the envelope parameters, and the overall volume. Then set the gate bit, leave it set for the length of the note, and reset the bit when the note is done. Multiple voice music is a matter of doing things in the right order and at the right times. Actual programming is a bit more complicated since individual bits must be changed without disturbing the others.

Filters

An audio filter functions in a way analogous to a paper coffee filter. The paper filter has little holes that will let the coffee pass through, but not the grounds. Make the holes a little bigger, and some of the finer grounds will pass through. The audio filter acts on frequencies and there are two basic kinds

of filters: *low-pass* and *high-pass*. A low-pass filter allows low-frequency waves to pass through, while the higher-frequency waves are blocked to a considerable extent. A high-pass filter rejects low-frequency waves and allows high-frequency waves to pass. The borderline is called the *cutoff frequency*. The C64's SID chip allows application of *high-pass*, *low-pass*, *band-pass* (rejects high and low), or any combination of the three. The cutoff frequency and *resonance*, or sharpness, of the filter can be selected. However, there is only one filter unit, which can be switched in or out for each voice, so you can't filter the voices differently.

If you set the cutoff frequency somewhere above the fundamental frequency and apply the filter in the low-pass mode, you will get a purer sounding, though softer, tone. The filter can also be applied in the high-pass mode to increase the harmonics, while attenuating (i.e., reducing the volume) of the fundamental.

Synchronization and Ring Modulation

These sophisticated controls can be used to produce complex waveforms. *Synchronization* causes the voice to lock onto the frequency of another voice. Voice 1 syncs with voice 3, voice 2 with voice 1, and voice 3 with voice 2. Depending on the two frequencies, interesting effects can be produced.

Ring modulation is the result of adding and subtracting two waveforms. When ring modulation is on for a particular voice, its output is the result of modulating from another oscillator. Non-harmonic frequencies (i.e., not multiples of the fundamental frequency) result, and very strange sounds can be produced. As its name implies, this can also be used to produce bell or chime effects. Voice 1 is modulated by voice 3, voice 2 by voice 1, and voice 3 by voice 2.

Voice 3 has extra controls available. Its output can be turned off — useful for synchronization and ring modulation used on voice 1 when you don't want to hear the driving frequency. In addition, the oscillator and envelope generator outputs are available in digital form in two SID registers. When random noise is selected as the waveform for voice 3, the oscillator output is a very good source of random numbers.

The sound output is available in a form compatible with good sound systems, so you don't have to rely on your TV's poorer system.

Programming C64's Sound

It is difficult for the BASIC programmer to take full advantage of all the SID's capabilities. Even such simple things as setting the gate bits require ANDing and ORing. Machine language is more effective when some of the more sophisticated features are involved.

In MICRO's Commodore 64 Data Sheet (MICRO 55:109) most of the SID's registers were inadvertently omitted. See page 9 in this issue for a complete list.

The *Commodore 64 User's Guide* and the *Programmer's Reference Guide* include a number of sample programs that demonstrate most of the C64's music capabilities. The software developer's kit I mentioned last month includes a SID monitor and a fancy music program. The monitor allows you to change the contents of just about every SID register, while you listen to the result. The music program has a number of pre-programmed pieces, from "Magic Trumpet" to a Bach chorale. You can select an "instrument" for each voice, the tempo, and a few other parameters. As the music plays, the tune is displayed as notes on a clef. The part of the program that lets you compose your own music is not very good. The notes are keyed in by their alphabetic representations. This might be OK for copying sheet music, but it's not very good for the trial-and-error composing most of us are likely to do. Abacus Software's "Synthy-64," reviewed below, makes programming of multi-voice music a lot easier.

A third type of music program is in-

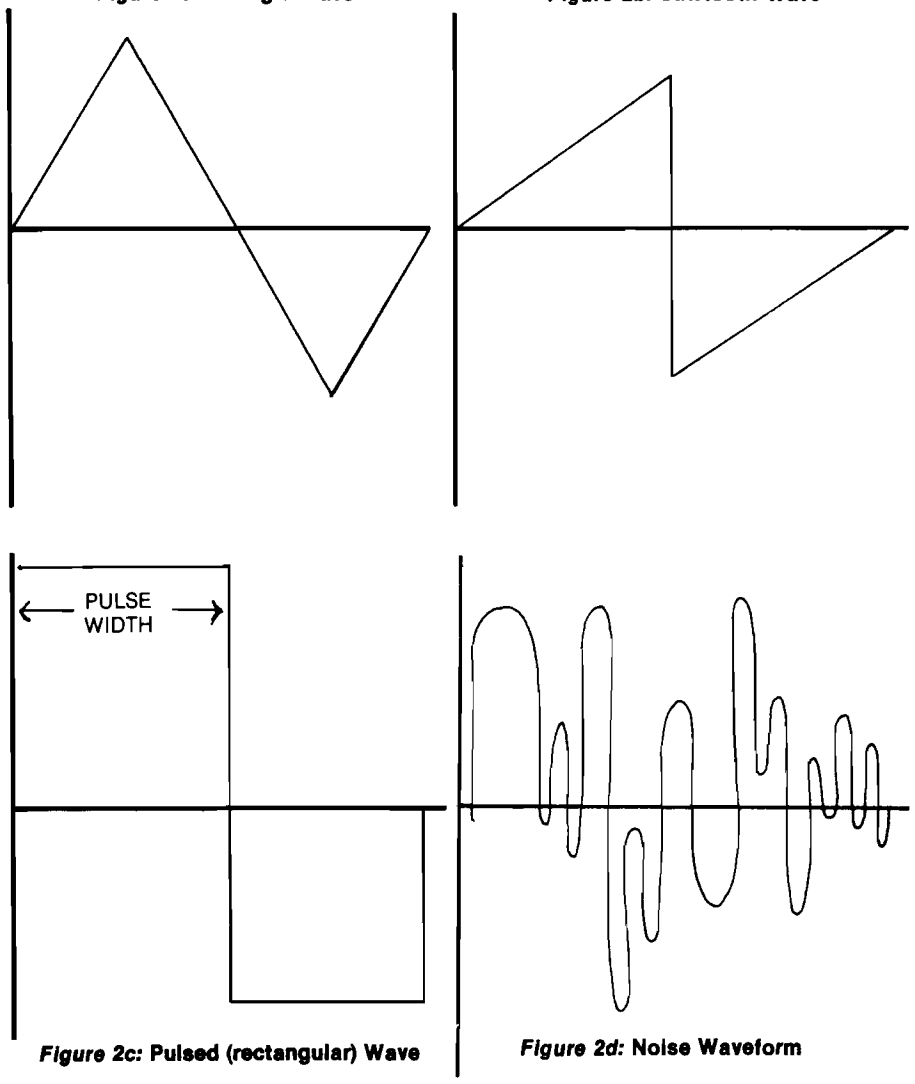


Figure 2a: **Triangle Wave**

Figure 2b: **Sawtooth Wave**

Figure 2c: **Pulsed (rectangular) Wave**

Figure 2d: **Noise Waveform**

teractive — one that allows you to play music in real time. The "Piano Keyboard" program in the user's manual is an example. It is impressive for what it does, but its limitations soon become apparent. Only one note can be played at a time. That's because the GET function can only read one key at a time. Also, it's easy to get ahead of the BASIC program. The effect is flattering for someone with uncoordinated fingers — the notes come out evenly spaced — but for others it is frustrating. A more complete interactive music program requires machine language for speed and should read the keyboard matrix to detect more than one key down at a time. Throw in a real musical keyboard, perhaps interfaced through the controller and parallel ports, and you would have something resembling a real synthesizer. Consider it a challenge!

Synthy-64: A Music Composition Program

Abacus Software's "Synthy-64" by Roy Wainwright is a music composition program that is considerably easier to use than the Commodore program described above, and a lot easier than figuring out all the POKEs yourself. Users familiar with musical notation will find Synthy-64's notation easy to grasp. Notes are entered with their alphabetic representations, followed by the octave number, a slash, and a number indicating the length (1 for whole, 2 for half, etc.). Octave numbers and durations remain the same, until changed, so shorthand notations can be used. Flats, sharps, naturals, key signatures, dotted notes, double-dotted notes, triplets, repeats, and rests are all easy to use. A sample program line is shown below.

```
10 SGN%2 B5/8 C/16 D E F G A6 B/2
```

Tempo and volume can be changed at any point during the composition. You can also program a "portamento" as if it were a regular musical note. A portamento is a continuous sweep, up or down, in the pitch of a note. The rate of sweep can be selected with a multiplier parameter.

The three voices are indicated with +, −, and £ prefixes. As with octaves and note durations, these designations are inherited by the next note, unless otherwise changed. The Synthy-64 interpreter has a read ahead feature that makes it easy to keep the three voices synchronized. There is no way to indicate measures, but this can be done by adopting the convention of using separate lines for each measure.

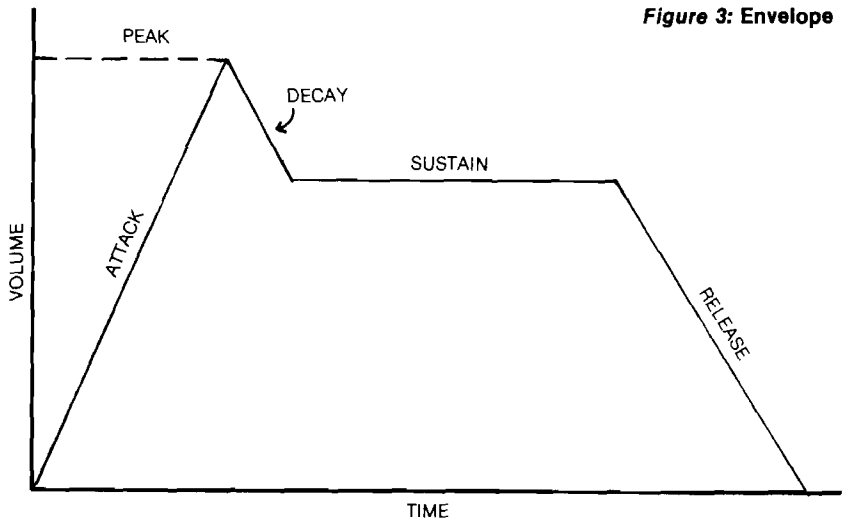


Figure 3: **Envelope**

That's all you really need to do some impressive three-part composing. The attack, decay, sustain, release, waveform, filters, etc., are set up with default values that produce a piano-like sound. In addition, the skeleton program includes convenient subroutines for flute, trumpet, banjo, accordian, and piano sounds. All of these registers, as well as ring modulation and synchronization, can be controlled with simple commands to set up your own sounds.

Synthy-64 is an interpreter that replaces the C64's BASIC interpreter, so most BASIC keywords don't work or work differently. The rudiments are there for control of program flow: GOTO, GOSUB, RETURN, STOP, END. The INPUT command is modified to handle multiple choice menus like a BASIC ON...GOTO. Text, including all C64 control and graphics characters, can be displayed simply by enclosing the characters in quotes. Just about everything else is missing, including the POKE statement needed to change screen or border colors (it works in immediate mode, but not within a program). LOAD and SAVE operations are implemented for both cassette and disk.

I found only two minor bugs in Synthy-64. The manual says you can tie a portamento to the previous note. My attempts to do this resulted in the interpreter skipping the portamento. Also described is a "skip ending" feature to repeat a phrase, skipping the rest of the program line, until a specified number of repeats have occurred. Instead, the remainder of the line is played the specified number of repeats, and then it is skipped — backwards! Synthy-64 changes octave designations between G and A, rather than the more conventional C split.

The manual is an adequate reference, although there are numerous typographical errors, a few of which result in misinformation.

Synthy-64 is, in general, a well-done, convenient music composition program. Nearly all of the C64's powerful sound features can be exploited. It is available for $35 from Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510.

MICRO'S VIC BOOK Available in March

I am currently editing, writing, and programming for MICRO's VIC book, to be released in March. Unlike most of MICRO's articles, the content will be aimed at the newcomer — the person who is just beginning to realize there is more to computing than running canned software. Eight or more useful and entertaining programs will be included on cassette and listed in the book, as well. The text that accompanies each program will aim first at using the program, then at making improvements by changing lines, and finally at understanding how the program works.

The most exciting program is one I call "MICRO Calc." MICRO Calc allows you to define elaborate mathematical calculations, make convenient changes, and calculate the results at the touch of a key. Screens can be saved on tape for future use. In addition the program can be used to learn how BASIC expressions work.

MICRO

68000 Program Control: Branch and Jump Instructions

by Joe Hootman

The 68000 installment this month covers the branch and jump instructions, both conditional and unconditional.

Branch and jump instructions allow the transfer of program control to another portion of the program. There are two basic types of program control instructions: the unconditional jump/branch and the conditional jump/branch. Table 1 catalogues the unconditional program control statements, and table 2 contains the conditional program control statements.

There are three basic types of unconditional program control instructions (table 1). The first type is the branch always (BRA) and jump (JMP). The BRA instruction branches with either an 8-bit or 16-bit displacement from the existing program counter contents. The JMP instruction jumps to an effective address expressed in the opword.

The BRA and JMP instructions do not save any return address information. If you want to return to the instruction following an unconditional branch, then you must Branch to Subroutine (BSR) or Jump to Subroutine (JSR). Both the BSR and JSR store the next instruction address on the stack before the next instruction is executed. The BSR instruction uses a specified displacement to designate the next instruction and the JSR instruction calculates an effective address to locate the next instruction to be executed.

The last type of unconditional branch is the Return and Restore CCR (RTR) and Return from Subroutine (RTS). These instructions are used at the end of a subroutine to return program control to the main program. A return is used when you want to continue execution at the original program location. When RTR is used, both the program counter and the CCR are

***Table 1:* Program Control Branch and Jump Instructions**

Mnemonic	Data Size/CCR	Name	Comments
BRA	8, 16 (offset displacement) CCR X N Z V C - - - - -	Branch Always	The program will always continue executing at the PC + displacement. The displacement is either 8 or 16 bits and is in two's complement form, the displacement is measured in the number of bytes. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 1 0 0 0 0 0 | 8-bit displacement 16-bit displacement The 16-bit displacement is zero if the 8-bit displacement is used.
BSR	8, 16 (offset displacement) CCR X N Z V C - - - - -	Branch to Subroutine	The address of the instruction immediately following BSR is stored on the system stack. The PC is loaded with PC + displacement. The offset displacement is either an 8-bit or 16-bit displacement and is expressed in two's complement form Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 1 0 0 0 0 1 | 8-bit displacement 16-bit displacement If the 8-bit displacement is zero, the 16-bit displacement must be used. Note a zero displacement cannot be used.
JMP	-- CCR X N Z V C - - - - -	Jump	The program will continue to execute at the address specified by the instruction. The address is specified by the addressing modes. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 1 1 | Effective Address: Mode | Register The effective address specifies the address of the next instruction. The following address modes cannot be used: 1, 2, 4, 5, 12, 13, 14.*
JSR	-- CCR X N Z V C - - - - -	Jump to Subroutine	The address of the instruction immediately following the JSR instruction is pushed onto the system stack and the program continues execution at the address specified. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 1 0 | Effective Address: Mode | Register The effective address mode specifies the location of the next location. The following address modes cannot be used: 1, 2, 4, 5, 12, 13, 14.*

(continued)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Name	Comments
RTR	-- CCR X N Z V C pulled from stack	Return and Restore Condition Codes	The PC and CCR are pulled from the stack and restored. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1
RTS	-- CCR X N Z V C	Return from Subroutine	The PC is pulled from the stack and the previous value of the PC is lost. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1
TAS	8 CCR X N Z V C - * * 0 0	Test and Set an Operand	The TAS instruction sets N if the most significant bit of the data is set, and clears N otherwise. The Z bit is set if the data is zero. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 0 1 0 1 1 | Effective Address Mode | Register The following address modes cannot be used: 2, 10, 11, 12, 13,14.*

*Addressing modes will be covered in future issues.

***Table 2:* Program Control Instruction (with tests)**

Mnemonic	Data Size/CCR	Name	Comments
BCC	8, 16 (offset displacement) CCR X N Z V C - - - - -	Branch Conditionally	If the specified conditions of the CC table are met, the PC will be loaded with the PC + (offset) and the execution of the program continued. The offset is specified in two's complement form, either 8 or 16 bits. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 1 0 | Condition | 8-bit displacement 16-bit displacement The 8-bit displacement is the two's complement specifying the relative distance between the branch and the instruction that is to be branched to. If the 8-bit displacement is zero, the 16-bit displacement is used.
DBCC	16 (offset displacement) CCR X N Z V C - - - - -	Test Condition Decrement and Branch	If the specified condition of the CC table for the loop is not true, the low order 16 bits of the counter data register are decremented by one. If the result is -1, the execution continues with the next instruction. If the result is not -1, execution continues at the current PC + the sign extended 16-bit offset displacement. The PC is the current instruction plus 2. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 1 | Condition | 1 1 0 0 1 | Register displacement The condition field is specified by the 16 bits in the CC table. The register field specifies the data register which is the counter. The displacement field specifies the distance of the branch.

(continued)

pulled from the stack. When RTS is used, only the progam counter is pulled from the stack.

The Test and Set an Operand (TAS) is designed to control the access of the 68000 to shared RAM. The TAS instruction is a read/modify/write instruction and tests the data to see if the most significant bit is set. The byte of data is returned to memory with the most significant bit set. The TAS is designed to allocate RAM to multiple users. If one of the multiple users wants to use a portion of RAM, he first reads the "TAS bit." If the most significant bit is set, the user knows that memory immediately adjacent to the TAS is being used. If a TAS byte has a zero in the most significant bit position, the user knows memory is available for use. The most significant bit is then set on the TAS byte and returned to memory to indicate that the memory immediately adjacent to that TAS is being used.

The conditional transfer of control from one part of a program to another is accomplished by testing one or more of the bits in the CCR. The conditional branches are given in table 2. The branch conditional instruction tests the bits in the CCR in accord with the CC table and branches if the conditions are met. The branch displacement is expressed as a two's complement 8-bit or 16-bit offset. If the 8-bit displacement is zero, the 16-bit displacement is used; if the 16-bit displacement is zero, the 8-bit displacement is used.

The Test Condition Decrement and Branch (DBCC) instruction is useful because it will count the number of times a specified condition in the CCR is not true. One of the data registers is used as a counter, and when the value in that register reaches -1, the instruction immediately following the DBCC instruction is executed. DBCC is useful for implementing a loop while a condition is true in the CCR.

The Set According to Conditions (SCC) is an instruction that tests the particular bits in CCR and sets the byte specified by the EA. If the condition is not true, the byte is set to zero.

STATISTICS

PURE AND SIMPLE

HUMAN SYSTEMS DYNAMICS

Human Systems Dynamics programs offer you flexibility, accuracy, and ease of use. You can purchase from the HSD statistics specialists with complete confidence. Any program that doesn't suit your needs can be returned within 10 days for full refund.

NEW

STATS PLUS $200.00

Complete General Statistics Package
Research Data Base Management
Design and Restructure Your Files
Count, Search, Sort, Review/Edit
Add, Delete, Merge Files
Compute Data Fields, Create Subfiles
Interface with other HSD programs
Produce Hi Res bargraphs, plots
1–5 way Crosstabulation
Descriptive Statistics for all Fields
Chi-Square, Fisher Exact, Signed Ranks
Mann–Whitney, Kruskal–Wallis, Rank Sum
Friedman Anova by Ranks
10 Data Transformations
Frequency Distribution
Correlation Matrix, 2 way Anova
r, Rho, Tau, Partial Correlation
3 Variable Regression, 3 t-Tests

ANOVA II $150.00

Complete Analysis of Variance Package
Analysis of Covariance, Randomized Designs
Repeated measures Designs, Split Plot Designs
1 to 5 Factors, 2 to 12 Levels Per Factor
Equal N or Unequal N, Anova Table
Descriptive Statistics, Marginol Means
Cell Sums of Squares, Data File Creation
Data Review/Edit, Data Transformations
File Combinations, All Interactions Tested
High Resolution Mean Plots, Bargraphs

HSD REGRESS $99.95

Complete Multiple Regression Analysis
Up to 25 Variables, 300 Cases/Variable
Correlation Matrices, Descriptive Statistics
Predicted & Residual Scores, File Creation
Regression on Any Subset of Variables
Regression on Any Order of Variables
Hi-Res Scatterplot & Residual Plot
Keyboard or Disk Data Input
Case x Case Voriable x Variable Input

Apple II, 48K 1 or 2 Disk Drives
3.3. DOS, ROM Applesoft

Call (213) 993-8536 to Order

or Write:
HUMAN SYSTEMS DYNAMICS
9249 Reseda Blvd., Suite 107
Northridge, CA 91324

VISA master charge

Table 2 *(continued)*

Mnemonic	Data Size/CCR	Name	Comments
SCC	8 CCR X N Z V C - - - - -	Set According to Condition	The specified condition of the CC table is tested. If the specified condition is true, the byte specified by the EA is set to all ones, otherwise the byte is set to all zeros. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 1 | Condition | 1 1 | Effective Address Mode | Register The condition field is specified by one of the 16 codes given by the CC table. The effective address field specifies the location of the T/F byte. The following address modes cannot be used: 2, 10, 11, 12, 13, 14.*

CC Table

	Name	Code	CCR Tested		Name	Code	CCR Tested
CC	Carry Clear	0100	$\overline{C}$	LS	Low or same	0011	$C + Z$
CS	Carry Set	0101	C	*LT	Less than	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
EQ	Equal	0111	Z	MI	Minus	1011	N
*GE	Greater or equal	1100	$N\cdot V+\overline{N}\cdot\overline{V}$	NE	Not equal	0110	$\overline{Z}$
*GT	Greater	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$	PL	Plus	1010	$\overline{N}$
HI	High	0010	$\overline{C}\cdot\overline{Z}$	*VC	Overflow clear	1000	$\overline{V}$
*LE	Less or equal	1111	$Z+\overline{N}\cdot\overline{V}+N\cdot V$	*VS	Overflow set	1001	V
**F	False	0001	0	**T	True	0000	1

* Conditions used with two's complement arithmetic.
** These tests are used for SCC, DBCC only.

MICRO™

C COMPILERS–COMMON FEATURES:
• UNIX VER 7 compatibility • standard float, double, and long support • run time library with full I/O and source • fast compilation and execution • full language.

AZTEC C II CP/M (MP/M) **$199**
• produces relocatable 8080 source code • assembler and linker supplied • optional M80 interface • SID/ZSID debugger interface • library utility • APPLE requires Z80 and 16K card

AZTEC C][APPLE DOS **$199**
• relocating assembler supplied • APPLE SHELL • VED editor • library and other utilities • requires 16K card

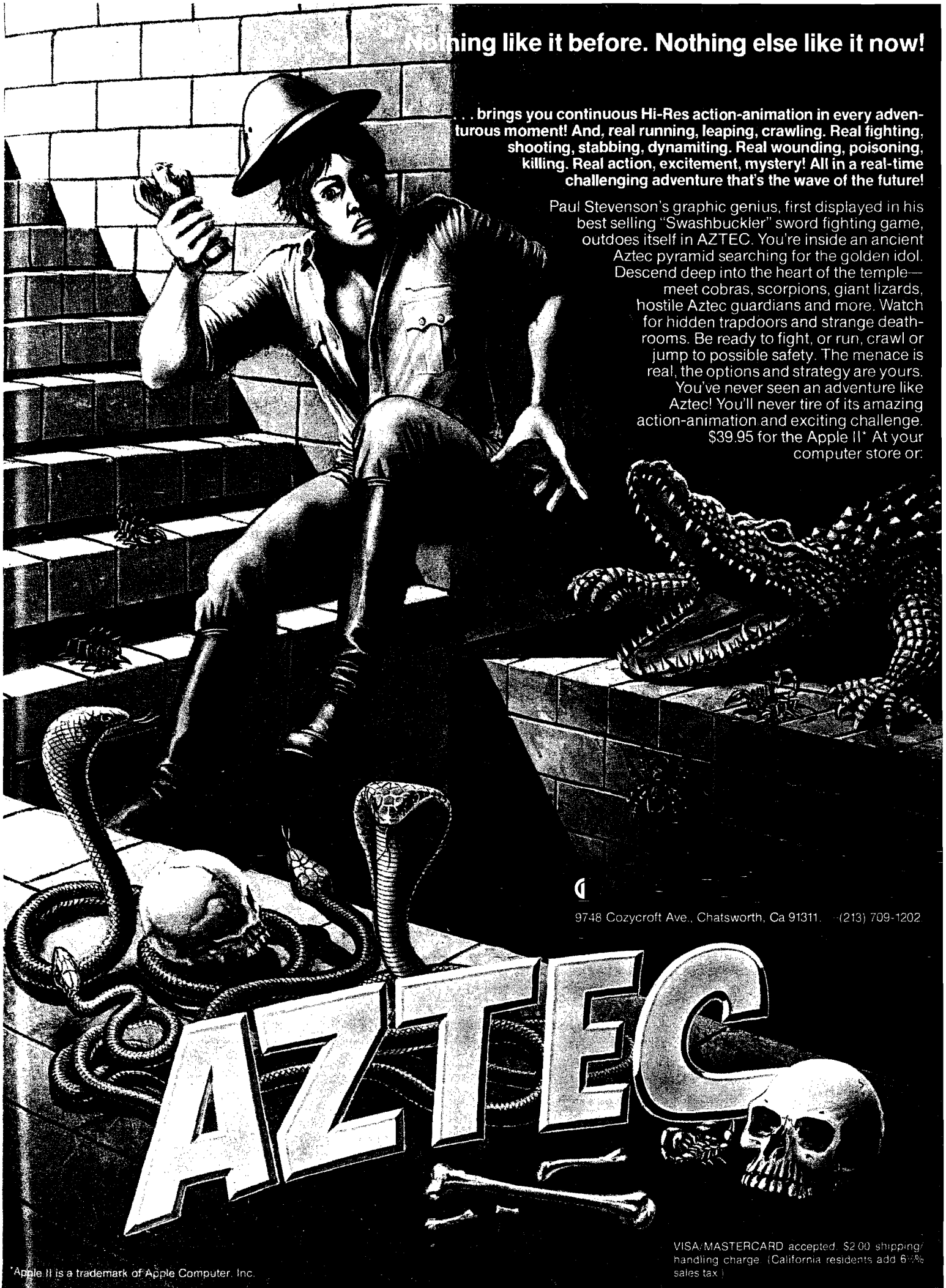
C86 IBM PC MSDOS CP/M-86 **$249**
• directly produces 8088/8086 object code • linker supplied

Manuals—$30 ORDER BY PHONE OR BY MAIL-Specify products and disk format

MANX software systems
Box 55, Shrewsbury, N.J. 07701 (201) 780-4004

master charge VISA

CP/M FORMATS: 8" STD HEATH, APPLE, OSBORNE, NORTHSTAR, OUTSIDE USA-Add $10 In N.J. add 5% sales tax


Nothing like it before. Nothing else like it now!
. . . brings you continuous Hi-Res action-animation in every adventurous moment! And, real running, leaping, crawling. Real fighting, shooting, stabbing, dynamiting. Real wounding, poisoning, killing. Real action, excitement, mystery! All in a real-time challenging adventure that's the wave of the future!
Paul Stevenson's graphic genius, first displayed in his best selling "Swashbuckler" sword fighting game, outdoes itself in AZTEC. You're inside an ancient Aztec pyramid searching for the golden idol. Descend deep into the heart of the temple—meet cobras, scorpions, giant lizards, hostile Aztec guardians and more. Watch for hidden trapdoors and strange death-rooms. Be ready to fight, or run, crawl or jump to possible safety. The menace is real, the options and strategy are yours. You've never seen an adventure like Aztec! You'll never tire of its amazing action-animation and exciting challenge. $39.95 for the Apple II* At your computer store or:
9748 Cozycroft Ave., Chatsworth, Ca 91311. (213) 709-1202
AZTEC
VISA/MASTERCARD accepted. $2.00 shipping/handling charge. (California residents add 6½% sales tax.)
*Apple II is a trademark of Apple Computer, Inc.

More on Tiny PILOT for the PET

by Arthur Hunkins

The author's observations include how to solve a few tricky problems, more detailed explanations of some commands, and how to convert the program for 8K PET.

As an educator, I appreciate PILOT as a simple, instructionally oriented, first language. Therefore, I was fascinated by the article, "Tiny PILOT for the PET" (MICRO 49:73), by Jim Strasma and John O'Hare. Since this PILOT required only 2K of memory, I couldn't wait to code it into my Upgrade 8K PET.

I quickly discovered, however, that Strasma and O'Hare's code was for 32K PET/CBM. Many changes are required to convert it to 8K. The modification, though time-consuming, is straightforward and easy to explain. It involves changing all numbers in the range $78-$7F to the range $18-$1F, and beginning the code at $1800. Exceptions to the number changes are these: do *not* change a number in the range of $78-$7F if it is followed immediately by a second number in the same range — change only the second number. For an example see the Tiny PILOT listing (MICRO 49:75): leave the value in $7846 as is and change the one in $7847. Also, three other locations should not be changed: $78AD, $7838, and $7F2E. That's it for the coding changes.

The initialization sequence for the 8K version becomes NEW:SYS7936 ($1F00 instead of $7F00). To run PILOT programs, do SYS6145, or use the ordinary RUN command.

This relocated Tiny PILOT runs on any PET/CBM with Upgrade ROMs and 8K or more of memory. The 6K not used by PILOT should be adequate for most PILOT programs. For the exceptional case, the conversion instructions above permit tailoring Tiny PILOT to any size memory on a 4K boundary. For example: on a 16K PET, convert $78-$7F to $38-$3F (with the exceptions mentioned), start coding at $3800, and initialize at $3F00.

The initialization routine protects PILOT one page lower than necessary in high memory, reserving 2.25K instead of 2K. If you need the additional page, change $7F10 and $7F11 (or $1F10 and $1F11 in the 8K adaptation) to $EA (NOP). You may alternately POKE7952,234 and POKE7953,234 in the 8K version or, after initializing, simply POKE53,24.

To understand Tiny PILOT better, the reader should consult the article containing the original Tiny PILOT, upon which Strasma and O'Hare based their version for the PET. "Tiny PILOT: An Educational Language for the 6502," by Nicholas Vrtis (MICRO 16:41), offers substantial and lucid details of user operation and internal programming logic.

One of the attractive features in the new implementation is the incorporation of graphics and cursor control features into the screen display; e.g., RVS field, CLR screen, and placement of text or graphics anywhere on the screen. In addition, there are special statements that reverse the field of everything currently displayed, and that scroll the screen up or down. The statements D: (Delay) and W: (Wait) also are particularly relevant to the instructional environment. Such educationally meaningful features in a 2K package are unique indeed. Tiny PILOT's statements are both highly practical and readily comprehended by the young computerist.

There are other user-oriented features. Hitting RETURN in response to I:, ?:, or A: statements gives either zero or a null string; it does not knock you out of the program as does BASIC's INPUT (press STOP to exit the program). Also, L:G-Z, unimplemented subroutines that might inadvertently find their way into a program, return appropriate ERROR messages — they do not bomb leaving the user buried in a machine-language crash.

Special challenges occur when you attempt to deal with string variables in Tiny PILOT. The language's greatest limitation — and the one that saves the most memory — is a near lack of string variable capability. (Some modest additional capacity in this area would be most welcome.) As it stands, the only viable string variable applications involve matching responses, and incorporating the user name string and strings converted from numerics in text. The only directly specified string, the name string $?, is entered by the user in response to a ?: statement. It is later referenced in Match and Type statements.

Numerics may be included in text only by first converting them into strings. This is accomplished with the Compute statement, C:$=N, where N is a previously defined numeric variable (see Sample Program, MICRO 49:74). The same conversion is required for Match, as matching is done only on text strings. Thus, 20, 2/5 and "2gether" all match with 2. This feature is awkward when the intent is to match numbers. One solution, again illustrated in the Sample Program is to subtract the numbers from one another, convert the result to a string, then Match to zero. In PILOT code these are: C:R=A-B, C:$=R, M:0, where A and B are the numbers being compared.

There are two important omissions in the list of PET Tiny PILOT Program Statements (MICRO 49:73). E:, listed

in the original Vrtis article but absent here, is an Exit from Subroutine used in conjunction with U: (Use Subroutine). The combination of U: and E: gives PILOT a highly viable subroutine capability. (By the way, E:, in contrast to BASIC's RETURN, is simply ignored when not executing a subroutine — a useful feature in some situations.)

The other error is in the fourth Program Statement: TEXT should read M:TEXT. This line describes the Match statement (also clarified by Vrtis). Note that, as mentioned above, Match operates only on text, and that multiple Matches are allowed per statement (commas separating the Match items). The Match option M: – checks for a negative "number" — evidently a minus sign as the first character. M: + is not implemented, but would be a welcome and memory-efficient addition.

You should be aware of several little quirks in the operation of Jump and Use Subroutine statements. With J:A (Jump to last Accept), you get an ERROR message if you attempt to jump back out of a subroutine you previously jumped into. J:A works fine as long as it is not inside a subroutine. Also be sure to reserve J:A for jumping to the last encountered A: (Accept) statement. "A" is a reserved label and any use of "A" as an ordinary label will result in an ERROR message. The referenced A: statement must already have been encountered in the program; jumping ahead to A: is not possible. Note, too, that U:A (Use Subroutine starting at last Accept) is not dependable. Such a branch can be accomplished by giving the Accept statement a regular label (such as *ZA:), and referencing it (U:Z).

I would like to suggest two additional enhancements to Tiny PILOT. First, since the P: statement generates only random integers from 1-99, getting random numbers in any other range is cumbersome (though possible). An optional extension to the routine, specifying a different range (for example, P:X,7 for a 1-7 span), would simplify coding. Second, a numerical Match statement (with positive and negative compare) also would help compact code.

In any case, if you have a PET/CBM with Upgrade ROMs, try Tiny PILOT. It's a most worthwhile educational investment, especially for the young computerist just beginning to program. And it won't cost you anything but time.

Editor's Note: "Tiny Pilot for PET" was published as object code only. Those wishing to have this program for 4.0 BASIC may want to obtain the source code from:

ASM/TED Users' Group
c/o Brent Anderson
200 S. Century St.
Rantoul, IL 61866

Arthur B. Hunkins may be contacted at the School of Music, UNC-G, Greensboro, NC 27412.

MICRO

COMPU SENSE

CENTIPOD **$27.95**
Like Centiped, only better!

FROGEE **$27.95**
The exciting arcade game of Frogger.

MOTOR MOUSE **$29.95**
What a cheese'ee game!

CRIBBAGE
VIC-20 **$14.95** C-64 **$17.95**
This is the game of Cribbage.

STAR TREK
VIC-20 **$12.95** C-64 **$17.95**
Excellent adventure game!

MASTER MIND
VIC-20 **$12.95** C-64 **$19.95**
Makes you think.

ROACH MOTEL **$9.95**
Kill the bugs!

YAHTZEE 1.1 **$12.95**
YAHTZEE 2.1 **$14.95**

GENERAL LEDGER **$19.95**
(VIC-20)

CHECK MINDER
VIC-20 **$19.95** C-64 **$24.95**

HOME INVENTORY **$19.95**
(VIC-20)

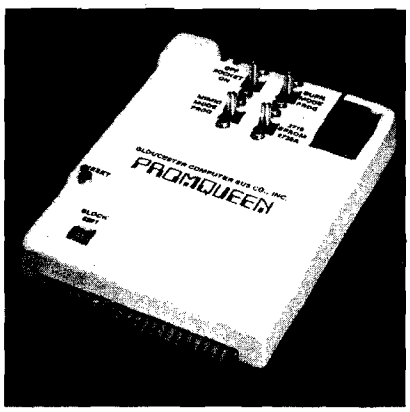
TO ORDER
P O BOX 18765
WICHITA, KS 67218
(316) 684-4660

Personal checks accepted
(Allow 3 weeks) or
C.O.D (Add $2.00)
Handling charges $2.00
VIC-20* is a registered trademark of Commodore

VISA MasterCard

VIC-20 USERS: Get Serious With A PROMQUEEN

- A cartridge development system
- Program from Commodore VIC-20 keyboard into built-in 4K ROM emulator
- Jumper to target ROM socket
- Test programs in circuit
- Built-in EPROM programmer and power supply
- Burns & runs EPROMS for the Commodore VIC-20, too
- Comprehensive manuals
- Fits EXPANSION PORT
- Includes Hexkit 1.Ø, a powerful 100% machine code editor/debugger utility program that makes coding for 8-bit Micros a snap.



Programs 2716, 2732, 2732A, 27C16, 27C32, adaptable to 2532 & 2764

PROMQUEEN CARTRIDGE COMPLETE ONLY $199

GLOUCESTER COMPUTER, INC.
One Blackburn Center
Gloucester, MA 01930
617-283-7719

Send for Free Brochure

VISA MasterCard

OFFICIALLY APPROVED BY COMMODORE

RH ELECTRONICS, INC.

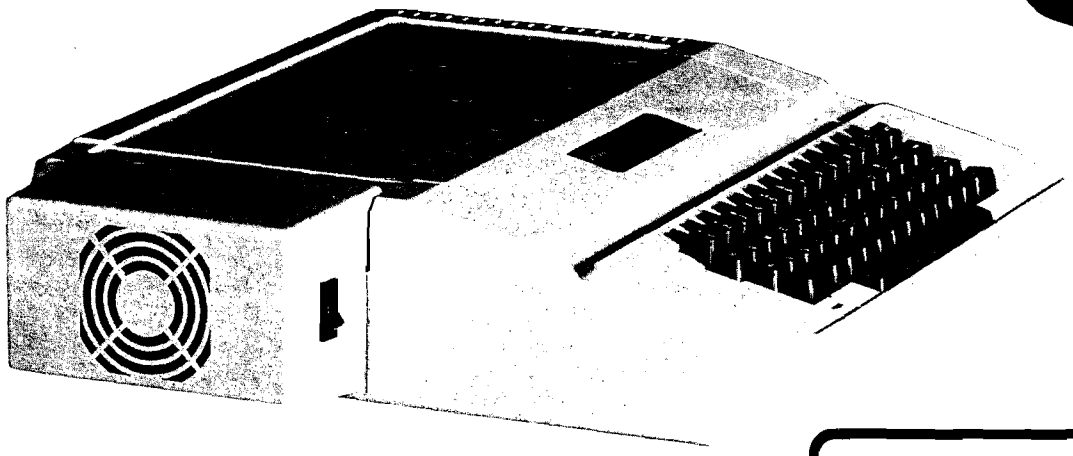
COMPUTER PRODUCTS
DESIGNING • MANUFACTURING
ELECTRONIC ENGINEERING

COPYRIGHT © 1981 — PATENTS PENDING

566 Irelan, Buellton, CA 93427
(805) 688-2047
8:00 TO 5:00 CALIFORNIA TIME

SUPER FAN II™

FOR YOUR APPLE II* COMPUTER

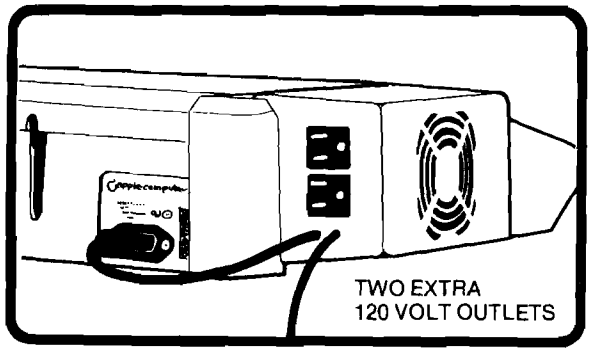


One Year Warranty

$74.95

With Zener Ray™ Protection $109.00

MASTERCARD — VISA



"COOL IT"

- ALSO FITS ON APPLE'S* NEW MONITOR STAND
- RED PILOT LIGHT ON/OFF SYSTEM SWITCH
- CLIPS ON — NO HOLES OR SCREWS • REPLACEABLE SWITCH
- AVAILABLE IN 120V or 240V AND 50/60 HZ • DURABLE MOTOR
- REDUCES HEAT CAUSED BY EXTRA PLUG-IN CARDS
- SOLD WORLD WIDE • UNIQUE 1 YEAR WARRANTY
- TAN OR BLACK COLOR • QUIETEST FAN ON THE MARKET
- INCREASED RELIABILITY — SAVES DOWN TIME AND REPAIR CHARGES
- LOW NOISE DUE TO DRAWING EFFECT OF AIR THROUGH YOUR COMPUTER AND SPECIAL FAN AND MOTOR DESIGN
- TWO EXTRA 120V OUTLETS FOR MONITOR AND ACCESSORIES TURN ON WHEN YOU TURN ON YOUR FAN (NOT AVAILABLE ON 240V MODEL)

SUPER FAN II™ WITH ZENER RAY OPTION $109.00

ZENER RAY™ TRANSIENT VOLTAGE SUPPRESSOR

OUR BUILT IN ADVANCED DESIGN UNIT GIVES
DRAMATIC COST SAVINGS — STOPS ANNOYING DOWN TIME
INSURANCE FROM VOLTAGE SPIKES - GLITCHES
DANGEROUS VOLTAGE SPIKES CAN JEOPARDIZE YOUR COMPUTER SYSTEMS
PROTECT COMPUTER - DISK DRIVE - PRINTER AND MONITOR

NO CUTTING WIRES • WON'T VOID WARRANTY, JUST PLUG IN SUPERFAN II WITH ZENER RAY

OTHER PRODUCTS BY RH ELECTRONICS, INC.

SUPER RAM II™ 16K RAM CARD FOR YOUR APPLE II. 2 YEAR WARRANTY $125

GUARDIAN ANGEL™ AN UNINTERRUPTABLE POWER SOURCE $595

12 VOLT TRANSVERTER 12 VOLT — RUNS YOUR APPLE II COMPUTER AND AND 5¼" DRIVE FROM YOUR CIGARETTE LIGHTER $149

*Registered trademarks of Apple Computer Inc.

DEALER INQUIRIES INVITED

A BASIC Match Routine for CAI

by Robert Phillips

Using a match routine, CAI authors can effectively trap errors, isolate mistakes, and overlook typographical errors. This article discusses some uses for a match routine, and presents such a routine in BASIC.

MATCH
requires:
A Microsoft BASIC

One of the nice things about a computer is its unfailing accuracy; yet one of the frustrating things about a computer is that same compulsive accuracy — it can't even recognize a simple typographical error. But users inevitably make typographical errors; also, they may make responses to questions in a program that the author didn't anticipate, such as "sure" instead of "yes." The programmer must try to overcome these problems. When he asks a "yes/no" question, for example, he may test the first letter of the response and if it is a 'Y' he can assume the answer is affirmative. He can use a statement such as

```
IF LEFT$(A$,1) = 'Y' THEN GOTO 999
```

and have the program branch appropriately, regardless of the answer (YES, Y, YEP, YEAH, etc.). However, when designing teaching materials for CAI (Computer-Assisted Instruction), the programmer may find himself facing difficulties because of typographical errors and misspellings. He must find a way to determine if the response *matches* what he is looking for. The purpose of this article is to show some ways in which a "match" routine can be used to write CAI effectively, and to present my MATCH algorithm, written in Applesoft and adaptable to most dialects of BASIC).

The Match Routine

Originally I wrote this algorithm in PL/1 for use on an IBM 360. Since I knew it worked there, I translated it from PL/1 into BASIC, rather than starting from scratch in BASIC. Some of the logic shows its PL/1 parentage. I followed IBM's method of using the match facility ("Partial Answer Processing") in their CAI language Coursewriter (the "*" will not match a blank character).

Since Applesoft BASIC does not have a built-in index or position function, I use the MID$ function to scan the string for the items to be matched. When an asterisk is found, its position is noted, and then it is replaced by a slash (/) in both the author's string and the student's response. This replacement effectively masks the character to be ignored by making it identical in both strings. Note that it is necessary to find the sections separated by ampersands before masking with the asterisks; yet, the asterisk means to ignore a character. This is the most difficult part of the algorithm. (For a completely different way of attacking this problem, see the discussion on a match algorithm in the publication *Frontend* from CONDUIT, University of Iowa.)

Take a closer look at how the routine works. There are four possibilities:

Listing 1 THE MATCH ROUTINE

```
100 REM)) MATCH ALGORITHM ((
102 R = 0: IF S$ = A$ THEN R = 1:RETURN
104 FOR Z1 = 1 TO LEN (A$):IF MID$(A$,Z1,1) = "&" THEN GOTO 112
106 NEXT Z1: Z2 = 1: Z1 = 0: Z1$ = A1$: Z2$ = S$: GOSUB 154
108 Z5 = LEN(Z1$): IF MID$(Z1$,1,Z5) = MID$(Z2$,1,Z5) AND LEN(Z2$) = Z5 THEN
    R = 1
110 RETURN
112 Z8 = 0: IF Z1 = 1 THEN A$ = MID$(A$,2)  GOTO 120
114 Z3$ = LEFT$(A$,Z1-1): Z4$ = LEFT$(S$,Z1-1): Z1$ = Z3$  Z2$ = Z4$: GOSUB
    154
116 IF Z1$ () Z2$ THEN RETURN
118 A$ = MID$(A$,Z1+1): S$ = MID$(S$,Z1+1)
120 FOR Z1 = 1 TO LEN(A$): IF MID$(A$,Z1,1) = "&" THEN Z3$ =
    MID$(A$,1,Z1-1): GOTO 126
122 NEXT Z1: Z1 = 0: IF LEN(A$) = 0 THEN R = 1: RETURN
124 Z1 = 0: Z3$ = A$: A$ = ""
126 FOR Z4 = 1 TO LEN(Z3$): IF MID$(Z3$,Z4,1) = "*" THEN GO TO 138
128 NEXT Z4: Z4 = 0: Z6 = Z1 - 1  IF Z1 = 0 THEN Z6 = LEN(Z3$)
```

1. there are no masking characters; 2. the only masking character is '*'; 3. the only masking character is '&'; or 4. both '*' and '&' are used as masking characters. The first two possibilities (no mask or only asterisk) are easy to check. First I check (line 102) to see if A$ and S$ are equal. If they aren't, line 104 looks for the position of the first ampersand. If none is found, line 106 invokes the asterisk mask subroutine in line 154. If any asterisks are present, they are masked there. When the subroutine returns to line 108, Z1$ and Z2$ are the masked versions of A$ and S$. They are tested for equality; if they are equal, R is set to 1. The match routine then returns. (You do not need to set R = 0; you do that when you enter the match routine, and set R = 1 only just before a RETURN. That way the "default" at return is 0.)

Lines 154-166 are the "asterisk subroutine," used to mask an asterisk. Z2 is a switch to skip line 156, which deletes leading blanks; this is necessary when ampersands are used also. Line 158 finds the position of the asterisk. If none is found, the subroutine returns. Line 162 checks to be sure that it is a non-blank character. If it is a blank, you know that the match has failed. You don't need to return to the match routine, so POP the calling address inside the match routine and RETURN to where the match routine was invoked.

If the character is not blank, lines 164-166 replace the asterisk with '/' in A$, put a slash in the same position in S$, and then go back to line 158 to find any other asterisks. If not, it RETURNS to the line that called it. The choice of the slash as the mask was purely arbitrary. I felt that it was unlikely to occur in things my students write. You can change it to something else, such as a control character or a non-printing character; e.g., CHR$(92), the backslash.

As I planned the match routine, I realized that the best way for me to proceed was to compare from the front to the first ampersand, make a new string starting after the ampersand, and compare to the next ampersand, etc. If any section cannot be found, the match has failed and the routine returns 0.

To make things as clear as possible, in the following discussion I call each part a 'segment.' I define segment as any part of A$ between two ampersands, or between the beginning and the first ampersand, or between the last ampersand and the end of the string. If a segment has one or more asterisks in it, I call each part separated by the asterisks a "subsegment." Thus, if A$="& MAN & WOMAN &", there are two segments ("MAN" and WOMAN"). It is the same if A$="MAN & WOMAN". If A$="& MAN & WOM*N &" then there are two segments, with the second consisting of two subsegments. When an ampersand is detected in A$, the routine works with these segments, trying to find each one. Naturally, if at any time the search for a segment fails, the routine does a RETURN.

Line 104 finds the location of the first ampersand and goes to line 112. If the ampersand is the first character in the string, there is nothing in front of it to check, so line 112 strips off the ampersand and goes to line 120. If the ampersand is not the first character, line 114 substrings off the first segment from both A$ and S$; these segments are assigned to both Z1$, Z3$ and to Z2$,Z4$, respectively. The routine then invokes the asterisk function with GOSUB 154. (Here you want any leading blanks stripped off, so you must turn off switch Z2.) After the asterisk mask has been applied, the substrings Z1$ and Z2$ are compared in line 116. If they are not equal, the match has failed and the routine does a RETURN.

If the first segment is successful, line 118 discards the first segment (and the ampersand) from A$ and S$. The program then checks (line 120) for the position of the next ampersand. If it finds one, control passes to line 126. If it doesn't, check the length of A$. If there's nothing left in it, it means you've checked the entire response without doing a RETURN; the match has been successful, so R = 1 and you RETURN. If there is still more of A$ left, it means that this is the last segment and it didn't end in an ampersand, so you have to look for an asterisk (line 126).

Listing 1 *(Continued)*

```
130 Z9 = LEN(Z3$): FOR Z7 = 1 TO LEN(S$)-Z9+1  IF MID$(S$,Z7,Z9) = Z3$ THEN
    GOTO 134
132 NEXT Z7: RETURN
134 IF Z7 < Z8 THEN RETURN
136 Z8 = Z7: GOTO 152
138 Z5$ = LEFT$(Z3$,Z4-1): Z6 = Z4 - 1: Z7 = 1
140 Z9 = LEN(Z5$): FOR Z7 = Z7 TO LEN(S$)-Z9: IF MID$(S$,Z7,Z9) = Z5$ THEN
    GOTO 144
142 NEXT Z7: RETURN
144 IF Z7 < Z8 THEN RETURN
146 Z8 = Z7: Z6 = Z1 - 1: Z4$ = MID$(S$,Z7,Z6): Z2 = 0: Z1$ = Z3$: Z2$ =
    Z4$: GOSUB 154: IF Z1$ <> Z2$ THEN Z7 = Z7 + 1  GOTO 140
148 Z5 = LEN(Z1$): IF MID$(Z1$,1,Z5) = MID$(Z2$,1,Z5) AND LEN(Z2$) < = Z5 +
    1 THEN GOTO 152
150 RETURN
152 A$ = MID$(A$,Z1+1)
154 IF Z2 = 0 THEN GOTO 158
156 IFLEFT$(Z2$,1) = " " THEN Z2$ = MID$(Z2$,2)  GOTO 156
158 FOR Z3 = 1 TO LEN(Z1$): IF MID$(Z1$,Z3,1) = "*" THEN GOTO 162
160 NEXTZ3: RETURN
162 IF MID$(Z2$,Z3,1) = " " THEN POP: RETURN
164 Z1$ = MID$(Z1$,1,Z3-1) + "/" + MID$(Z1$,Z3+1)
166 Z2$ = MID$(Z2$,1,Z3-1) + "/" + MID$(Z2$,Z3+1): GOTO 158
```

Listing 2 TO INVOKE AND TEST THE MATCH ROUTINE

```
10 HOME: VTAB 4: PRINT TAB (4); "MATCH ROUTINE INPUT ALGORITHM": PRINT:
   PRINT TAB(9); "(USING BASIC ROUTINE)": VTAB 12
20 PRINT: INPUT "INPUT 'AUTHOR' STRING --) ";A$: IF NOT LEN(A$) THEN A$ =
   OL$: GOTO 50
30 TEXT: VTAB 8: CALL -958: PRINT "A$ = '"; A$; "'": POKE 34,10: HOME
40 OL$ = A$: IF A$ = "END" THEN TEXT: HOME: VTAB 24: END
50 PRINT: INPUT "'STUDENT' STRING --) "; S$: S$ = " " + S$ + " "
60 GOSUB100: PRINT: IF NOT R THEN PRINT "NO ";
70 PRINT "MATCH": PRINT: GOTO 20
```

The coding that starts on line 126 checks any segment that had an ampersand in front of it. This means you have to ignore as much of S$ as necessary to find the segment specified in A$. Before you scan for the whole segment, however, determine if there are any asterisks present and, if so, mask them. If line 126 finds an asterisk, the program goes to line 138. Here, S$ is scanned (line 140) for the substring in front of the asterisk. If it is not found, line 142 does a RETURN with no match. Line 144 checks to make sure that the segment found is in the string later than the last segment found. If both of these are OK, then the program substrings off the entire segment (this is easy: since asterisks represent one byte, you know how long the segment is and can use a MID$ to get the entire segment). You then GOSUB to the asterisk subroutine. If that doesn't match, it does not necessarily mean that there is no match; it means that you must scan further to see if you can find another occurrence of the subsegment in front of the asterisk. To do this, line 146 does a GOTO 140. If you get to line 148, it means you have found the right segment and have masked the asterisks. Check it. If it is good, go back to line 120 to find the next segment; if the comparison is false, you don't have a match, and so you RETURN.

Please note that there are lots of string operations in this routine, and string operations create garbage. If there are hundreds of string variables (as in an array, for example) in a large program that uses lots of memory, memory will need to be cleaned quite often, which may consume a considerable amount of time. In most circumstances, however, this should not present a problem.

If you are interested in developing uses for this routine, I suggest you use a very short program to get the strings called A$ and S$, and see if it accurately reports match/no match. Listing 2 is just such a short test program. You can put it in front of the Match routine to test it. In this test program, if you want to use the same author string again, press return without entering anything. To end, simply enter 'END' as the author string. Try it with representative samples to see if it is accurate and fast enough for your purposes.

Limitations and Modifications

When using this match routine in my programs, I have found no cases of the routine reporting incorrect results. However, experimentation has shown that the routine will give an incorrect result if a word with an asterisk in it is repeated and separated by ampersands, such as

& GRE*N & GRE*N &

This will report a match if it finds just one occurrence of the string. If the items are separated, such as

& GRE*N & MACHINE & GRE*N &

it always reports no match. These problems are caused by the need to check that the segments are in the right order. They are the only limitations in the algorithm of which I am aware. There may be others.

As indicated above, I followed the lead of IBM's Coursewriter in not allowing the asterisk to mask a blank. If you would like the asterisk to ignore a blank as well as any other character, change statement 162 to REM.

When items are separated by the ampersand, they must be in the order that they appear in A$. If order is unimportant,so that

& COMPUTER & MACHINE &

should give a match no matter which order the two words are in, change statements 134 and 144 to REM. I have not tested the accuracy of all possible combinations when lines 134, 144, and 162 are all chanaged to REM.

Uses for the Match Routine

There are various types of CAI. One is "drill and practice," in which the student is given quick, rapid-fire drill on items that have one right answer. Another type is called "tutorial," which serves as a tutor to the student, giving practice but also supplying explanations to the student. If the CAI program is to act as an effective tutor rather than a drill-master, it is necessary to identify student errors and diagnose them. Diagnostics explain the specific error to the student so he can understand the error, learn from it, and — hopefully — not repeat it.

The author who designs materials with diagnostics has two basic choices for formatting the student's response: some type of "objective" (or "controlled") questions, such as true/false, multiple choice, yes/no, etc.; or some type of open-ended (or "non-controlled") item. The controlled type is comparatively easy to program, although it does take time and care to develop effective materials. But some types of teaching activities simply cannot be done by a controlled-question format. As a foreign-language teacher, for example, I want my students to use the language forms actively, rather than merely to identify the correct form in a controlled format.

With non-controlled items, a student may be asked a question to which he responds with a sentence. It is impossible for an author to predict every possible variation of vocabulary, expression, and spelling that the student might use in his answer. Most CAI programs scan the answer, looking for a key word or two. If the student has used

those words, the answer is considered correct. For this reason, all the CAI languages with which I am familiar have some type of facility that enables the author to scan the student response.

What a Match Routine Does

A match routine can be used for several purposes: to scan for a key-word or key phrase, to overlook misspellings, and to detect what type of error a student has made, which is necessary for effective diagnostics. To do these things, the author must be able to tell the routine what to look for and what to ignore (selective ignoring or masking is what a match routine is all about). I use two characters (as in IBM's Coursewriter): the asterisk and the ampersand. The asterisk ('*') indicates a "single-character ignore;" that is, when scanning to see if the student's response matches the author's expectations, the routine ignores the character in the position of the asterisk. Thus, if the author asks to match COMPUT*R, the routine reports a match if the student response is "computer" or "computor" or "computir" or even "computzr". Because the asterisk ignores only non-blank characters, a response such as "comput r" will yield a no match. The ampersand ("&'), which is a "multiple-character ignore," means ignore anything and everything (or even nothing!) in that position. It works just like the 'wild-card' character '=' used in some utility programs, such as FID. Thus, an instruction such as

```
& COMPUTER &
```

tells the routine to report a match if it finds the word "computer" anywhere in the response.

The asterisk and ampersand can be used together so that an instruction such as

```
& COMPUT*R &
```

returns a match if it finds the word "computer" or "computor" or "computir" or even "computzr" anywhere in the student response. Note that the placement of the ampersands may be crucial. If there is not an ampersand at the beginning, the word must be the first word in the response, since the routine is not instructed to ignore anything in front of the first word. Similarly, if there is no ampersand at the end, the word will have to be the last! Several different words can be used, separated by ampersands:

```
& COMPUTER & MACHINE &
```

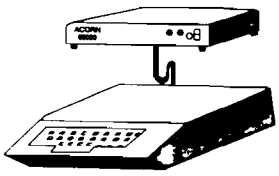
will give a match only if the student uses both words and in that order. Some things that match are "The computer is a useful machine," "My computer is bigger than a sewing machine," etc. However, "A computing machine" would give a no match ('computer' is not there) as would "That machine is my computer" (the words are out of order). Figure 1 shows more examples of strings that match or don't match.

One use for a match routine is to find a key word or a key phrase. To do this, the author puts the key word or key phrase between ampersands. If it is important that they be stand-alone words, rather than parts of words, there must be a space between the ampersands and the word; if not, then the ampersands should not be separated.

A second use for a match routine is to be able to ignore misspellings and typographical errors. To do this, the author may substitute the asterisk for the letter(s) most likely to be misspelled. Or, the author can use the ampersand

ACORN 68000

ATTACHED PROCESSOR FOR THE APPLE II™



$1495

HARDWARE

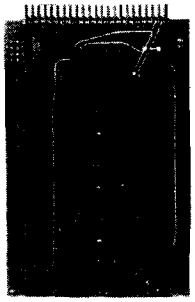
- 68000 Microcomputer with 12 MHZ clock
- 131,072 Bytes of RAM Memory
- 32,768 Bytes of ROM Memory
- Two RS 232c serial ports up to 19,200 bps
- One million bps interface with APPLE™
- Seven levels of vectored interrupts
- Real time clock and timer
- Separate case and power supply
- Uses only one peripheral slot in the APPLE™
- Invisible operation with APPLESOFT or PASCAL
- Compatible with Compilers and 6502 Assembly Programs
- 68000 Assembly Language Development System

Write or call for a free brochure or send $10 for 100 page users manual (refunded with order for ACORN)

ACORN SYSTEMS INC.
4455 TORRANCE BLVD., #108 • TORRANCE, CA 90503
Telephone (213) 371-6307

*Apple, Apple II and Applesoft are the trademarks of Apple Computer Co.

32K CMOS STATIC RAM BOARD for SYM/AIM
Models MB-132/32K, $299
/16K $241,/8K $197

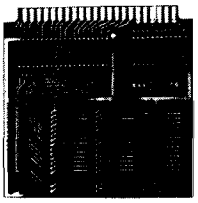


Features:

- 200ns Low Power CMOS, STATIC RAM
- Extends your expansion connector
- Plug compatible with 2716 EPROMS
- First 8K are jumper selectable
- Entire board may be bank-switched
- G-10 Glass epoxy, Full solder mask, Gold fingers
- Full 1-year limited warranty

I/O EXPANSION BOARD for the SYM/AIM
I/OX-122 $60
I/OX-222 $72

and other microcomputers that use 6522 VIAs for I/O and do not provide full address decoding on board. This board has physical space for four additional 6522 VIAs, and provides additional decoding for a total of 16 devices. Connectors for all I/O lines, and further expansion are included. All 6522 functions are available, with no interference with previous functions of the original VIA. Two versions of this board are available. The I/OX-122 mounts above, and directly plugs into, an on-board 6522 socket, and relocates the original VIA to the expansion board. Where there are space limitations, the I/OX-222 uses a dip header and an 8" cable for remote installation.



REAL-TIME CLOCK/CALENDAR $60 Write for Info.
P.O. Box 1019 • Whittier, CA 90609 • (213) 941-1383

ALTERNATIVE ENERGY PRODUCTS

to ignore several characters. Of course, the two can be used together.

A third use for a match routine is to find out exactly where a student has made a mistake. In my opinion, the most effective CAI does not merely tell the students that they have made a mistake, but it diagnoses the error; i.e., it tells them what they did wrong and offers an explanation. The only way an author can effectively diagnose errors is to know what the error is. Diagnosis is comparatively easy with controlled items; with non-controlled items it becomes more difficult — precisely when diagnosis is most important.

This match routine is not very fast since it is in BASIC rather than machine language. Putting some Spanish CAI on my Apple, I found that it takes about 1.4 seconds to compare this author's string

& LE DECIMOS & QUE & SE ACUESTE &
(we tell him to go to bed) — put English translation here, please, author!)

with this student response: "Nosotros le decimos a Juan que se acueste temprano" [We tell John to go to bed early]. That is not too long, but if the student makes some unanticipated mistake, it takes the program even longer to work its way through to find the error. (I have timed the program to take as long as five seconds in involved cases where the match routine must be invoked many times to analyze one sentence). To obscure this time lapse, at various stages I include progress messages such as "THE SENTENCE IS INCORRECT" when the right answer is not matched, "FIRST VERB IS CORRECT" as the checking continues, etc. The length of time is unnoticeable since the processing is completed before the student finishes reading the messages.

Linking the Match Routine To a Program

I use the match algorithm as a subroutine in my BASIC programs, accessing it with a GOSUB. I start it on line 100 (see the Match Routine), extending it to line 166. (My input routine with pre-processor is the only thing earlier in the program. Since both are accessed frequently, I put them as close to the top as possible.) Because you cannot pass arguments, you must make assignment statements before the GOSUB statement. The routine expects the string variable A$ (for 'Author') to have the string to be matched; the variable S$ (for 'Student') is assigned the student response (after pre-processing eliminates punctuation, compresses double blanks, and puts a blank at each end). I usually combine everything on one line:

```
A$ = "& KNI*ES &" : S$ = IN$ :
GOSUB100
```

The match routine reports the result *via* the variable R: R = 0 if there is no match, R = 1 if there is, and destroys the strings A$ and S$ during processing. The match routine needs to use intermediate strings, and some index and pointer variables. The following variable names are used: Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8, Z9, Z1$, Z2$, Z3$, Z4$, Z5$. Any values stored in these variables when the match routine is invoked may well be destroyed.

Figure 1: **Examples of "Match" and "No Match"**

A$	S$	Match?
& COMPUT*R &	COMPUTER	Yes
	COMPUTIR	Yes
	COMPUTING	No
	COMPUTE OR DIE	No
& COMPUT&	COMPUTER	Yes
	COMPUTE OR DIE	Yes
	COMPUTING MACHINE	Yes
& GREEN &	GREEN FIELDS	Yes
	GREENSLEEVES	No
	GREAT SPLEEN	No
& GRE*N &	GREEN BEANS	Yes
	A GREEN AND RED COLOR	Yes
	IT IS GREAN	Yes
	GRAIN	No
& GR&N &	GREEN FIELDS	Yes
	GREENSLEEVES	No
	GREAT SPLEEN	Yes
	GROWN MAN	Yes
	GROANING	No
&HOW&	I DON'T KNOW HOW TO DO IT	Yes
	HE WENT TO THE SHOW	Yes
	MITCH OWES ME MONEY	No
& B*T&	BYTE	Yes
	BIT	Yes
	BOUGHT	No
	I BAT THIRD	Yes
& COMPUT& MACHINE &	A COMPUTING MACHINE	Yes
	A COMPUTER OR A MACHINE	Yes
	THE MACHINE IS A COMPUTER	No
	COMPUTING MACHINERY	No
	COMPUTE THE COST OF THE MACHINE	Yes

Conclusion

In applications where users will be responding with words and phrases, a routine that masks part of the input often is necessary to analyze that input. The routine may scan for key words, overlook spelling and typographical errors, or find exactly where an error occurs. For whatever purpose, the two masking characters (the asterisk and the ampersand) enable the author to use and analyze verbal input intelligently to achieve the intended results. This match routine is one tool that can be used to do that.

Robert Phillips has B.A., M.A., and Ph.D. degrees in Spanish. He is a professor at Miami University, Ohio, and is Assistant Chairman of the Department of Spanish and Portuguese. He has been working with Computer-Assisted Foreign Language Instruction since 1970 and has written CAI in PL/1, FORTRAN, APL, Coursewriter, and BASIC. You may contact Professor Phillips at the Dept. of Spanish and Portuguese, Miami University, Oxford, OH 45056.

An Overview of Educational Software

by George Gerhold

As a follow up to our October education issue, this article offers a discussion on specialized systems designed for educational applications.

Educational software can be classified into three groups: applications software — programs designed to help the users (students) to master a particular body of material; general systems — standard packages useful in an educational setting; and educational systems software — specialized languages and systems designed for educational applications. Our emphasis here will be on educational systems software.

But let's begin with a few comments about applications software. There are vast amounts of such material in existence (something like 10,000 hours worth for the PLATO system alone). Much of the applications software for microcomputers is of very poor quality, much of it is hard to locate (it was written by a teacher for use in a particular classroom, and a half-hearted attempt at marketing was made), and most of it is poorly documented. A number of publications and organizations have attempted to address these problems. The Microsoft Project (Northwest Regional Educational Laboratory) is the most ambitious attempt, in particular their effort to establish a data base that lists sources and reviews. It appears that there is an obvious need for quality control *via* reviews, but there are many problems. Applications software is supposed to provide individualized instruction, and what works well in one setting with one group of students and one teacher may fail in another setting. How does one review truly individualized material?

Reading reviews of applications software is somewhat like reading reviews of recordings of contemporary classical music. Since there is no standard of excellence for comparison, the reviews may tell more about the reviewer's bias than about the quality of the product. A number of institutions are trying to cope with the problem by assembling libraries for preview of programs.

There are obvious problems with guaranteeing respect of copyright. Also, many of these institutions have spent their whole budget on hardware and are relying on donated software. We estimate that it will take close to 1,000 such centers to adequately cover the country. No small supplier of software will be able to supply that number of free copies! A number of the textbook publishers have entered the educational software field. Their products tend to place heavy emphasis on the most routine kinds of drill (heavy use of multiple choice or numerical answers) with elaborate record keeping. These programs are protected in ways that also prevent teachers from customizing them for their own classrooms. Time will tell whether or not the publishers will find a way to combine their mass marketing approach with the individualization that characterizes the best educational applications software.

Next we turn to general systems software with educational applications. Word processing systems have obvious educational utility. Current practice in composition courses is to emphasize the rewriting/editing process as the part of composition that can be taught. There are a number of schools using word processing software in this way. One of the problems is that many of the word processing packages are designed for commercial use and as such are more complex than necessary for student use. A student version of Wordstar would be a worthwhile product. (*Editor's note:* Memory Bank has just released "The Bank Street Writer," a word processor for students.) Unfortunately, much of the hardware being purchased by schools is not ideal for word processing because of short lines and fuzzy characters. Data base management systems could be widely used in schools, both in individual classrooms and in central offices. Again, simplified versions of business systems should find a market here. At present there are a number of administrative packages on the market; for example, gradebook, attendance, and library packages. One of the most impressive is the Harts III package that, in addition to the items listed, also handles class scheduling for a large (1,000 students) school. Without doubt, the most widely used packages in this class are general language interpreters and compilers used in programming courses. One worthy of mention is the Interpas-RASCAL system, an interpretative version of Pascal plus a set of graphic and sound programs. This system comes with a complete curriculum including textbooks and student and teachers' guides. It is currently available from McGraw-Hill (Gregg Division) for the Apple II. No doubt other similar packages will appear shortly. One final item likely to find wide acceptance is the disk library management system for keeping track of programs. Disk Master for the Apple II is a fine example of this type of program.

Finally we come to systems software designed specifically for educational applications. Again this is conveniently divided into three classes: programming languages, authoring languages, and authoring systems. LOGO is really the only language designed for student programming. LOGO offers many advantages for this purpose. It is highly structured and allows long variable names and procedure names. It uses advanced techniques like recursion in an elementary and natural way. Above all, its orientation is primarily graphic, at least at the beginning levels.

Graphic exercises offer many advantages for use with beginners. The appeal is sufficient to hold their interest, and debugging is a visual process rather than an abstract reasoning process. For example, "the program went wrong after drawing four line segments" is much easier to detect than "the program gave this wrong numerical answer by going off after the fourth numerical step."

LOGO is available for three different microcomputer systems, with more rumored to be on the way. Three different sources offer versions for a 64K Apple II with one disk; these versions differ only in minor ways. Texas Instruments offers a ROM version for the 99/4 with memory expansion. Radio Shack offers both a ROM version (16K) and a disk version (32K) for the Color Computer. There are significant differences between the three systems.

The Apple versions are probably closest to the original mainframe versions of LOGO. This is accomplished at the cost of speed, expensive hardware (remember 64K), and severe restrictions on the amount of user program space. The Texas Instrument version adds more colors and sprites, which aid animation. This version offers only integer arithmetic and draws lines only by character definition, an approach that noticeably limits the complexity of figures that can be drawn. The Radio Shack version requires the least hardware and adds multiple turtles. This latter feature allows use of LOGO for illustration of true multi-tasking and other advanced concepts. It also provides a way for doing simple animation. The Radio Shack version offers only integer arithmetic and eliminates all of the word and list processing operations of the original LOGO language. These differences reflect different analyses of what the educational applications of LOGO will finally be. Rumor indicates that there will be a sprite version for the Apple (requiring extra hardware) and a real number and line drawing version for the Texas Instrument. In considering LOGO as an educational tool remember that it has been available for only one year. No doubt there will be many new applications of LOGO in the near future.

Author languages are general computer languages designed to make the programming of instructional dialogs easier. Author languages reflect a different set of priorities than more familiar computation languages like BASIC and Pascal. For example, an author language makes programming the recognition of keywords in a response relatively easy, even when those keywords are misspelled by the user, but an author language has little need for the nested loop construct essential in number-crunching applications. Although there are a number of author languages for large computers, only PILOT has been implemented on a range of microcomputers. There are many versions of PILOT around, but the commercial products come from three sources. The original PILOT came from the San Francisco Medical Center. Nevada PILOT and Atari PILOT are close to the original version, but Atari PILOT has added turtle graphics to the package. A major set of additions to the original language were developed at Western Washington University. Micropi offers versions based on those extensions for CP/M, 6809, TERAK, IBM, Pascal machines, and for a number of larger mainframes. Apple PILOT, TI PILOT, Monroe PILOT, Color PILOT (Radio Shack), and the forthcoming PILOT for the Commodore 64 are all based on the Micropi version. IIAT offers a version of PILOT for CP/M machines.

Author languages offer several advantages for educational programming.

UPGRADE YOUR AIM-65* INSTANTLY

*A trademark of Rockwell Inc.

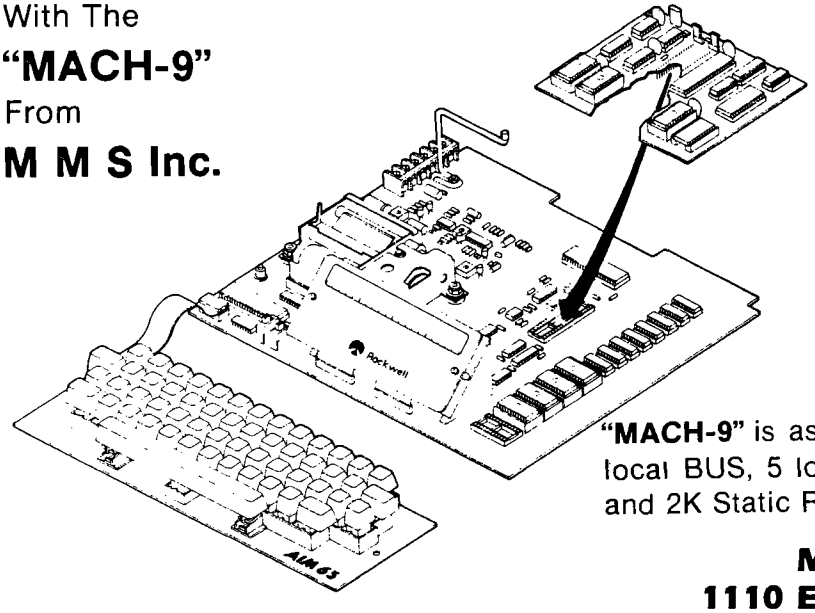
To A 6809 Development System

With The

"MACH-9"

From

M M S Inc.



INTRODUCTORY PRICE

$239.

Plus $6 U.P.S.
And Handling

Includes:

*6809 CPU Plug-in Assembly
*Super-set of AIM Monitor
*Two-Pass Symbolic Assembler
*Complete Monitor Source Listings
*Enhanced Cut & Paste Editor
*200 Page Manual
*Full I/O Control

"MACH-9" is assembled and tested with local BUS, 5 locking low force ROM sockets and 2K Static RAM

M M S Inc.
1110 E. Pennsylvania St.
Tucson, AZ 85714
(602) 746-0418

master charge VISA

Programmer (or author) productivity is much higher when using author languages. In fact programmer productivity when using a language like BASIC for such applications is often so low that they are forced to adopt multiple choice or numerical formats almost exclusively. Thus, a related benefit of using an author language is an increase in quality made possible by the increase in author productivity. Another advantage is that a flexible instructional program written in an author language is likely to be much more intelligible to teachers than the same program written in BASIC. This is important because the best educational software is under revision — based on student reactions — for many cycles. Intelligible programs get revised and improved; others do not. One of the assumptions behind the PILOT language is that it is easier to teach an experienced teacher how to program using an author language than it is to teach a computer programmer how to design good instructional material. Experience has shown that this is true. Now the problem is to get the teachers enough free time to actually do it!

Authoring systems represent another attempt to make it easier for experienced teachers to generate instructional software. Authoring systems present a menu of formats for instruction; for example, true-false, multiple-choice, and fill-in-the-blank. Once the teacher has selected the format, the authoring system builds the instructional program from the teacher's responses to questions. Typically the authoring system would pose a series of questions like: what instructions do you want to give the student, what is the first question you want to ask the student, what is the correct answer, what response do you want to give to a correct answer, where should the student be sent after a correct answer, what is a common incorrect answer, etc. In some authoring systems the teacher's input is encoded into a file; in others the teacher's input actually generates a PILOT-like program. There are a variety of authoring systems on the market. All of them work on only one machine so there is no transferability possible. Without question the most elaborate is Bell and Howell's PAS system. PAS provides the widest range of formats and ways to include true flexibility in answer processing, although the latter requires something very close to programming. PAS also provides a variety of type styles and some graphics. PAS also carries what must be the record price for microcomputer software — $15,000 educational or $20,000 commercial, plus a healthy yearly fee. If I were the instructor, I'd rather keep the money and learn to program!

Finally we come to the area where educational software is weakest — graphic design. One problem is that the people writing educational software have no training in graphic design, so they tend to ignore it. The other problem is that the software for graphic design included in most development packages is comparatively crude. Computer languages often draw on common knowledge to make learning computer languages easier. For example, many languages use an algebra-like syntax in numeric assignment. The problem is that we have no generally accepted language for graphics that can be translated into computer terms. Both software and hardware (e.g., graphics tablets) solutions have been tried, but much remains to be done. The next development in educational software tools probably will be in the areas of graphics and speech generation.

MICRO™

SCIENTIFIC SOFTWARE ASSOCIATES, LTD.

Q-card

Questionnaire Analysis Software

- Microcomputer based
  Avoid the expense of contract services -- do everything in-house on your own Apple II+ microcomputer.
- Easy data entry
  Avoid time consuming keypunching. Uses respondent-marked cards entered with an Optical Mark Reader (keyboard entry also possible).
- Comprehensive data analysis
  Sort on any variable(s), tally all responses, conduct cross tabs, correlations, linear regression, frequency distributions, and more.
- Complete editing capabilities
  Weight items, derive composites, add or delete items, and more.
- Easy-to-use
  Programs are user friendly, menu driven, and interactive. No special computer expertise is required.

Call or send for more information today.

SCIENTIFIC SOFTWARE ASSOCIATES, LTD.

BOX 208 · WAUSAU, WI. 54401

TELEPHONE: (715) 845-2066

Apple II+ is a registered trademark of Apple Computer, Inc.

MICRObits

Deadline for MICRObits: 20th of second month before publication; i.e., February 20th for April issue. Send typewritten copy (40-word limit) with $25.00 per insertion. (Subscribers: first ad at $10.00.)

Lessons in Algebra

An easy and fun way to learn the basic elements of high school algebra. Apple computer diskette $29.95. 30-day money-back guarantee if not satisfied.

George Earl
1302 So. General McMullen Dr.
San Antonio, TX 78237

Dynamite PET/CBM Accessories!

Write-protect switches/indicators for 2040/4040 disk drives. Real world software at low cost. 2114 RAM adapter (replaces obsolete 6550's) and 4K memory expansion for "old" 8K PETs. Hundreds of satisfied customers. Write for *free* catalog!

Optimized Data Systems
Dept. M, Box 595
Placentia, CA 92670

Double Precision Pascal

*Tera*Comp — A scientific mathematics library for Apple Pascal. Gives the Apple mainframe accuracy with 64-bit double-precision mathematics. Package includes matrix operations including inverse, double-precision trigonometric functions, and dynamic array allocation. $95.00 from:

PicoTera Systems
P.O. Box 1631
Corvallis, OR 97339
(503) 754-0237

MICRObits *continued on page 93*

Microcomputers in a College Teaching Laboratory, Part 4

by Deborah Graves, Richard H. Heist, Thor Olsen, Howard Saltsburg

Interfacing a microcomputer to two types of scientific instruments is described. These applications illustrate the importance of commonality in both hardware and software design. Interfacing a microcomputer to a scientific instrument can enhance the capabilities of the instrument, as illustrated with a spectrophotometer, or it can facilitate data reduction and increase productivity, as illustrated with a gas chromatograph.

In Parts I - 111 of this series (MICRO 53:53, 55:59, 56:38) we have described how the microcomputer is used for both data acquisition and process control in the undergraduate chemical engineering laboratory.[1-3] So far we have concentrated on interfacing microcomputers directly to laboratory experiments and replacing conventional analog instrumentation with a combination of A/D converter, microcomputer, and printer — our universal instrument.

A related area of application for the microcomputer involves interfacing to more sophisticated scientific instruments. There are two primary reasons for using a computer with a scientific instrument. The first is to enhance the capability of the instrument, and the second is to improve operational features, such as ease and speed of operation and data reduction. The gas chromatograph and the spectrophotometer are good candidates for microcomputer interfacing. Both find widespread application in chemistry and related fields, and all but the most expensive models require the user to spend a considerable amount of time reducing data.

The Gas Chromatograph

One of the most widely used analytical tools is the gas chromatograph (GC). The chemical engineering department is no exception, as we make heavy use of the GC in both undergraduate and research laboratories. In the undergraduate laboratory, it is not uncommon for students to generate 15 to 20 chromatograms in an afternoon. Each of these must be analyzed and the data processed to reach the objective of the experiment. The data analysis can be quite tedious and time consuming. If a strip chart recorder is used to record the data, retention time and peak areas must be measured with conventional methods. Also, data reduction will generally be done outside the laboratory and usually too late to repeat any analyses that yield questionable results. These features of the laboratory are very unattractive to the student and tend to obscure the important features of the experiments.

These time-related problems have been eliminated by using the microcomputer to acquire and process the data from GC analyses on line. Data reduction that would normally require hours is now done in a matter of minutes. Consequently, our students are now able to devote more of their time to the underlying conceptual aspects of the laboratory experiments.

The hardware required to interface the microcomputer to the GC is simple. The detector output is a voltage in the low millivolt range, similar to the output from thermocouples, as discussed in Part II of this series.[2] Consequently, the A/D-converter interface (QM-100 and signal amplifier) used for temperature measurements should be directly applicable. Initially our chief concern was sampling speed since it is possible to get rapidly changing signals (narrow signal peaks) from the GC. However, the QM-100 A/D converter allows sampling rates up to 50 Hz, and this has proven to be more than adequate for all our applications. The only remaining problem was to develop a computer program that would satisfy our needs and be easy for the students to use.

The software package developed for the GC system consists of both machine-language and BASIC programs. The machine-language program acquires and tests the data, and the BASIC program interacts with the user and performs data reduction. The user interface is designed to be friendly. The user doesn't need to be aware of the machine-language program since the BASIC program provides all instructions necessary to operate the system, as well as a menu of available options for data output. A user does not need a computer background to run the system.

To operate the system, the user injects a sample into the GC and produces a small pressure spike by momentarily interrupting the carrier gas flow. This spike produces a small output signal that causes the computer to begin the timing for the analysis and to look for incoming data. As the components of the sample pass over the detector, the computer stores the digitized signals and indicates to the user that it is accepting data.

After the data acquisition is completed, the user signals the computer to begin the data analysis. The program computes retention times and peak areas. If there is peak overlap the peak areas are resolved by dropping a perpendicular between the two adjacent peaks. After a short time (10 to 60 seconds, depending on the size of the data set) the user is queried as to which results he would like to see. One option is to display the spectrum, retention times, and peak areas on the computer

screen. Another is to generate high-resolution hard copy on a printer. The screen plot is generated using the 16 PET/CBM graphic characters that combine quarter-cursor elements. The linear resolution of the screen plot is thus twice that obtainable by simply plotting with full size characters. The screen plot allows previewing of data prior to plotting on the printer. See figure 1 for examples of the printer output.

The GC-microcomputer combination has been used successfully for over two years and has had a significant impact on the laboratory program. It has provided a convenient method of data reduction and demonstrated the utility of the microcomputer as a laboratory tool. In addition, by streamlining the experiments that rely on the GC for chemical analyses, the microcomputer-GC system has improved our ability to handle large numbers of students in the laboratory program. This point has been most important since increased enrollments have had a profound impact on our laboratory operation.

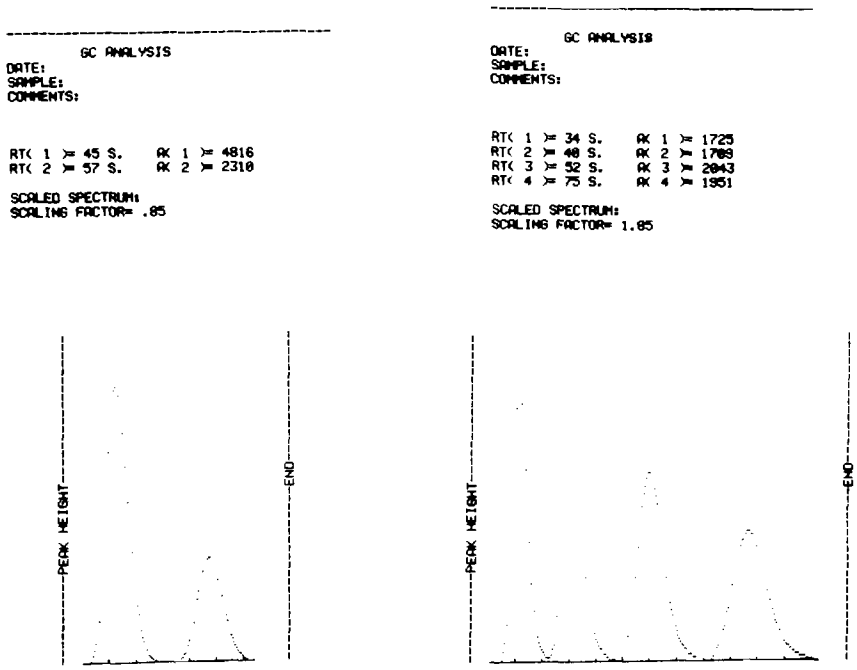


Figure 1: **High-resolution hard copy from two gas chromatography experiments. The first illustrates the chromatograph of a two-component mixture, and the second, a four-component mixture. The retention times and peak areas are indicated at the top of each plot. The chromatograms were obtained with a Carle Model 8700 gas chromatograph interfaced to a PET/CBM microcomputer and Trendcom 100 dot-matrix printer, as described in the text.**

The Spectrophotometer

Another important, commonly used analytical instrument is the spectrophotometer. Our department uses spectrophotometers primarily as research tools, although they have been used in the undergraduate laboratories to a limited extent.

The spectrophotometer presents the user with problems similar to those of the GC. The measurement of an optical absorption spectrum produces a lot of information that must be analyzed and then converted to another form (such as absorbence, transmittance, or extinction) to be of direct use. If the spectral scan covers a wide range of wavelengths, data reduction can be quite time consuming. As with the GC, there are expensive accessories that will do most of the work, but these devices are not available in all laboratories and certainly not in most teaching laboratories.

Most spectrophotometers use a photocell or a photomultiplier tube as a detection device, so the output signal is a current. Normally this is passed either to a chart recorder or a display of some sort, which converts the signal to a numeric representation. Since the output current of the detector can easily be converted to a voltage (see Part II of this series), it is a simple matter to use the QM-100 A/D converter and an amplifier to interface the spectrophotometer to the microcomputer. In this case, as with the GC, the hardware requirements are simple. The only significant problem was the development of the necessary software.

The software requirements for data acquisition with a spectrophotometer-computer system are not very different from those already discussed for the GC. The computer must be signaled when data collection is to begin, and it has to store the acquired data for processing when the analytical scan is finished. It is necessary to keep track of the time and the wavelength scan rate so the wavelength scale can be calibrated properly. Another requirement is that high-resolution hard copy of the absorption spectrum must be available for further analysis.

The BASIC portion of the program package, which interfaces with the user and performs the calculations, was specific to this application and had to be written in its entirety. Because of the similarity in the requirements of the data acquisition routine, however, it was possible to modify the machine-language code from the GC program (by deleting parts specific to the GC and adding a few routines for handling the timing) and apply it to the spectrophotometer system. This commonality among applications in both software and hardware requirements is important. It occurs frequently, and recognizing it can save a great deal of time and effort.

Since the spectrophotometer generally scans a wavelength range during an analysis, it is important to coordinate the sampling rate and the wavelength scan rate. We chose to use a timer available on one of the input/output chips (6522) of the PET/ CBM computer. With this timer it is easy to measure time intervals precisely and have the microprocessor interrupted at the end of each interval so that it can sample the spectrophotometer output. By knowing the scan rate and keeping track of the number of sample points collected, the wavelength for each point can be determined. Once the computer has collected the data, absorbence or transmittance is easily calculated. Details are available in the literature concerning the PET/CBM input/output ports[4, 5] and the 6522.[6]

One very significant advantage of this system over the use of a chart recorder is that the data can easily be stored, either internally or on an external mass storage device. Therefore, it is a simple matter to convert a single-

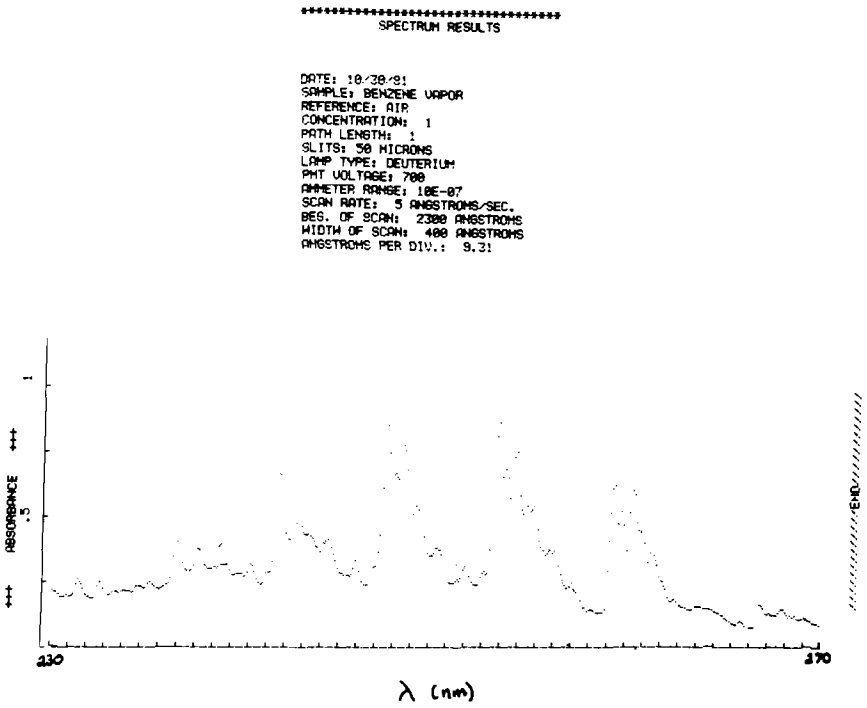


***Figure 2:* The ultraviolet absorption spectrum of benzene vapor at room temperature. The spectrum was obtained using a GCA McPherson single-beam spectrophotometer interfaced to a PET/CBM microcomputer and Trendcom 100 dot-matrix printer.**

beam spectrophotometer into what is effectively a dual-beam device. The basic requirement is that the analysis be run twice, first in the reference mode, and then in the sample mode. An example of this type of application is shown in figure 2. This absorption spectrum of benzene was taken at room temperature with a single beam GCA McPherson spectrophotometer. The units of the wavelength scale are 4.56 Angstroms/division; the ordinate is in absorbence units. The high spectral resolution is evident and comparable to that obtained from much higher-priced instruments. Individual portions of an absorption spectrum can be selected and magnified by changing the scan rate and the wavelength range and repeating the scan. Dual-beam spectrophotometers are generally more desirable than single beam devices, but they are also much more expensive. Thus, the interfacing of a microcomputer to the spectrophotometer resulted in enhanced capability without significant additional expense.

References

1. Saltsburg, H., Heist, R.H., and Olsen, T., *Microcomputers in a College Teaching Laboratory — Part 1*, MICRO, No.53, October, 1982, pg. 53.
2. Heist, R.H., Olsen, T., and Saltsburg, H., *Microcomputers in a College Teaching Laboratory — Part 2*, MICRO, No. 55, December, 1982, pg. 59.
3. Olsen, T., Saltsburg, H., and Heist, R.H., *Microcomputers in a College Teaching Laboratory — Part 3*, MICRO, No. 56, January, 1983, pg.
4. West, R.C., *Programming the PET/CBM, The Reference Encyclopedia for Commodore PET/CBM Users*, Compute! Books, Greensboro, N.C., 1982.
5. Hampshire, N., *The PET Revealed*, Computabits Ltd., Somerset, England, 1980.
6. See, for example, DeJong, M.L., *Programming and Interfacing the 6502, with Experiments*, Howard W. Sams & Co., Inc., Indianapolis, 1980; Zaks, R., *6502 Applications Book*, Sybex, Berkeley, 1979.

You may contact the authors at the Department of Chemical Engineering, University of Rochester, Rochester, NY 14627.

MICRO™

computer case company

• AP103

Attache-style cases for carrying and protecting your complete computer set-up. Accommodates equipment in a fully operational configuration. Never a need to remove equipment from case. Simply remove lid, connect power, and operate.

AP101	Apple II with Single Drive	$109
AP102	Apple II with Two Disk Drives	119
AP103	Apple II, 9 Inch Monitor & Two Drives	129
AP104	Apple III, Two Drives & Silentype Printer	139
AP105	13" Monitor with Accessories	99
AP106	AMDEK Color Monitor	119
RS201	TRS-80 Model I, Expansion Unit & Drives	109
RS204	TRS-80 Model III	129
AT301	ATARI Computers with Peripherals	109
P402	Centronics 730/737 & Radio Shack Printer	89
P403	Epson MX70/80 or Microline 82A	89
P404	Epson MX100 Printer	99
P405	IDS 560 or Prism 132 Printer	109
P406	Starwriter/Printmaster F-10 Printer	119
P407	Okidata Microline 83A or 84 Printer	99
P408	Prowriter 2 Printer	99
P409	Prowriter (Apple Dot Matrix) Printer	89
IB501	IBM Personal Computer	129
IB502	IBM Monitor	99
HP601	HP41 with Accessories	99
CM703	Commodore Model 64 with Drives	119
CM704	Commodore Model 64 with Dataset	109
NS010	North Star Advantage	139
CC80	Matching Attache Case (5")	85
CC90	Matching Attache Case (3")	75
CC91	Matching Accessory Case	95
CC92	5.25" Diskette Case	49

computer case company

5650 Indian Mound Court
Columbus, Ohio 43213
(614) 868-9464

CALL TOLL FREE
800-848-7548

MICRO™

CoCo Bits

By John Steiner

This month's column looks at techniques that allow you to interface your machine-language routines with BASIC. The Color Computer memory map, published by Tandy, leaves many undocumented locations. Hopefully I can put a few items in their correct places.

To demonstrate the use of a RAM hook, I have included a list-pager program. (More on this later.) If you know any locations of RAM hooks, or addresses of particular BASIC functions in ROM, please contact me.

Ben Farmer of Charlottesville, VA, sent the following information on a print-routine hook: locations 159, 160, and 161 are called during print to screen or printer; and locations 410 to 412 seem to hook to the keyboard after each BASIC keyword. Mr. Farmer also points out that there is a compatibility problem with EDTASM+ and the 8-bit printer driver distributed by Radio Shack to people with 1.0 ROM. The driver is required to work with the DMP-100 printer, yet it won't work with the assembler. If anyone has found a solution to the problem, let me know.

While I am on the subject of RAM hooks, I have a routine that interfaces with the LIST and LLIST command to page a list on the screen (see listing 1). Before loading the program, enter CLEAR 200, &H7FE5. This protects the routine from BASIC. Load the program by reassembling or POKEing the data into memory. To activate the routine, the hook at locations 383, 384, and 385 is used. These locations contain an RTS, and 383 is accessed after a LIST or LLIST. If an instruction is found, it can be executed. In this case, a JMP to the routine will be executed. As assembled, the page lister is written to fit at the top of a 32K machine. With one exception, the program is written in position-independent code. If you move the program, the location LINCNT must be defined to a valid RAM location.

The program lists 14 lines to the screen and stops, waiting for a keypress, after which the next 14 lines are listed. To hook the program to BASIC, after loading the routine, enter:

```
POKE 383, &H7E : POKE 384, &H7F :
POKE 385, &HE7
```

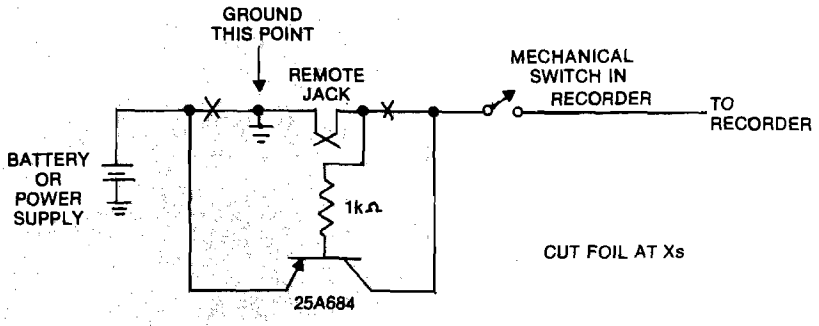
If you do not have extended BASIC, you will have to convert the hex numbers to decimal. Load a BASIC program and do a LIST. (If you do an LLIST, you will find that the program does not stop after 14 lines, but continues until completed.)

The routine is called just before each line in a listing. The opcode $7E causes a JMP to the start of the program at $7FE7. The routine checks location $006F (DEVNUM), which determines printer or screen status. If $6F contains $FE, the list is going to the printer; a $00 indicates the screen is the destination of the print. When anything but zero is found, control returns to BASIC. If screen printing is being done, $7FE6 is incremented by 1. A CMP tests to see whether or not the fourteenth line has been printed. If not, control returns to BASIC. You can change the number of lines printed by changing the data in $7FF2.

When 14 lines have been printed, control jumps to the GETKEY routine in BASIC. The program loops while waiting for a keypress, after which the count is reset to zero and control returns to BASIC.

Other applications for this particular RAM hook come to mind. For example, it would be easy to supply a line feed after the BASIC-generated carriage return for those printers that need it.

Bob Gilbertson of Fargo, ND, provided me with the circuit in figure 1. This circuit is a modification that allows the connection of high-current-drain cassette recorders to the light-duty relay inside CoCo. If you use a standard Radio Shack series recorder, you don't need this modification; but if you can't bear to spend money on a new recorder while an older machine just sits on the shelf, you can use this circuit. The RS recorder has a very light current drain, and the relay provided is more than adequate. If you use a recorder with too much current drain,




```
                        00100 *PAGER PROGRAM
                        00110 *JOHN STEINER
                        00120 *NOV. 8, 1982
                        00130       ORG $7FE6
                        00140 LINCNT EQU $7FE6
                        00150 DEVNUM EQU $6F
7FE6      00            00160       FCB $00
7FE7 0D   6F            00170 START TST DEVNUM    CK FOR LLIST
7FE9 26   14            00180       BNE OUT       IF SO THEN GO
7FEB 7C   7FE6          00190 LININC INC LINCNT   ADD 1 TO LINE
7FEE B6   7FE6          00200       LDA LINCNT
7FF1 81   0E            00210       CMPA #$0E     SCREEN FULL?
7FF3 26   0A            00220       BNE OUT       IF NOT THEN GO
7FF5 AD   A000          00230 LOOP  JSR [$A000]   GO GET KEYPRESS
7FF9 4D                 00240       TSTA          CHECK FOR KEY
7FFA 27   F9            00250       BEQ LOOP      IF NO KEY THEN WAIT
7FFC 7F   7FE6          00260 RESET CLR LINCNT    SET LINE COUNT TO ZERO
7FFF 39                 00270 OUT   RTS
```

CoCo Bits *(continued)*

the relay might stick closed and the recorder will not stop at the end of SAVEs and LOADs.

Bob's circuit places a PNP transistor in series with the power supply. The computer relay switches the transistor, which in turn switches the recorder. The circuit is representative of most portable cassette recorders. All wiring indicated exists within the cassette recorder itself; only the transistor and 1K resistor are added to the circuit. The Xs indicate where a wire or foil should be cut. The jack assembly indicated is the motor remote control jack. Cut the lines as shown and install the transistor and resistor. (If your cassette recorder is positive ground instead of negative ground, as in the schematic, use a suitable NPN transistor.)

The circuit works by current flowing through the base lead, base bias resistor, and remote relay switch inside the computer. Although a transistor number is specified, almost any general-purpose PNP transistor will work. Choose one with a collector current at least twice the normal current draw of your particular cassette recorder. The switching current is quite small and will keep the relay from sticking closed.

When you finish the modification, you can test its performance with the following information. BASIC has two routines in ROM that control the condition of the cassette motor: MOTOROFF is located at $A7E9, and MOTORON is located at $A7CA. EXEC &HA7CA turns on the motor, and EXEC &HA7E9 turns it off. Another way to control the relay is with a POKE to the PIA at 65313. POKE 65313, 60 turns on the motor, and POKE 65313, 52 turns if off. Of course, you could just type MOTORON or MOTOROFF, but that wouldn't be nearly as much fun.

Next month, I will take a look at some more RAM hooks and present more CoCo news. If you have anything to contribute, contact me at the address below, or through MICRO.

You may contact the author at 508 Fourth Avenue NW, Riverside, ND 58078.

MICRO™

MICRObits *(Continued from page 88)*

Target-an AIM 65 Newsletter

Need information for your AIM 65 computer? News, software, and hardware are examples of items covered in the newsletter. Yearly subscription rates are $7.00 in the US and Canada, $12.00 elsewhere. Back issues are available beginning with 1979 at the same per year rate.

Target
c/o Donald Clem
RR#2
Spencerville, OH 45887

The State of the Art in Astro-Software

Wide range of astrological and astronomical software of the highest quality. From powerful (and income-producing) astrological charting service packages and printing interpretation packages, to super-accurate computer ephemerises. For all Commodore computers, Apple II Plus, and TRS-80.

Matrix Software
315 Marion Avenue
Big Rapids, MI 49307

OSI Super Defender

Play this great arcade game at home. All machine code includes: scanner, smart bombs, laser fire, moving mountains, and more. Save your humanoids from the alien landers. Very smooth (half-character moves) graphics. $14.95 for C1, 2, 4 tape or 5¼" disk.

DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

Commodore-64 Software!

Enjoy these excellent professionally written programs: *Septa-Cube Logic Puzzle* — 3-dimensional; combine 7 pieces into a solid 3 × 3 × 3 cube. *Advanced Sound Synthesizer* — with graphics. Each program $17.95 cassette, $21.95 disk. Add $2.00 postage/handling charge. Send for *free* catalog.

Dynamic MicroGraphic Software
4289 Union Rd.
Buffalo, NY 14225

MICRO™

VIC-20

VIC-20 INTERFACING BLUE BOOK

Did you know that your VIC can be used to control a 99¢ toy motor so effectively that it runs like a precision machine? Or that you can build an accurate digital thermometer using the VIC and four parts costing less than $5?

These and other 18 interfacing projects selected for usefulness, ease of construction and low cost are detailed in the VIC-20 Interfacing Blue Book, a veritable gold mine of practical information on how to build a variety of interfaces for your computer.

Projects include: Connecting VIC to your stereo; Pickproof digital lock; Capacitance meter; Liquid level sensor; Telephone dialer; Voice output; 8K/16K RAM/ROM expansion; 128K RAM expansion; 8-bit precision D/A; 8-bit A/D converter; MX-80 interface and more.

Written by a college professor in a friendly and informative style, the Blue Book gives you theory of operation, schematics, program listings, parts list, construction hints and sources of materials for each one of the 20 projects.

If you want to get the most out of your VIC this book is a must. Cost is $14.95 (less than 75¢ per project!). Price includes postage.

VIC-20 VIC-20 VIC-20 VIC-20 VIC-20 VIC-20 VIC-20 VIC-20 VIC-20 VIC-20

microsignal Dept N
P.O. BOX 22
MILLWOOD NY 10546

Please send me a copy of the Blue Book.
Enclosed my check for $____

NAME _____

ADDRESS _____

Above prices include postage in the U.S. CA res. add 6% tax. Foreign add $2.

COMPU SENSE

"CARD/?"
(CARD/PRINT)

UNIVERSAL CENTRONICS PARALLEL PRINTER INTERFACE FOR THE VIC-20®

Now you can use your VIC-20® with an EPSON MX-80 printer, or an OKI-DATA printer, or a TANDY printer, or just about anybody's printer. And you don't have to give up the use of your user port (MODEM), or change to special printer commands, or load any special software driver programs to do it.

- Outputs standard ASCII codes to the printer.
- Plugs in the VIC-20® printer serial i/o port.
- Understands all standard VIC-20® print commands.
- No modification to your VIC-20®.
- No special programs required.
- Includes all necessary cables to hook up a standard printer using centronics parallel input.
- MADE IN THE U.S.A.

The "CARD/?" is a product of CARDCO, Inc.

$79.95

TO ORDER.
P. O. BOX 18765
WICHITA, KS 67218
(316) 684-4660

VISA
MasterCard

Personal checks accepted
(Allow 3 weeks) or
C.O.D. (Add $2.00)
Handling charges $2.00
VIC-20® is a registered trademark of Commodore

HUNTINGTON COMPUTING

Softlights

By Fred Huntington

Time to tell you about a nifty little program that's been around a while that deserves some mention. The Menu Generator is an excellent program that everyone should use on every disk to create a HELLO program to take the typing out of running programs.

It creates menus quickly and painlessly. The publisher has even given permission to programmers to use the generated menus in commercial programs — no royalties. Comes complete with a free backup disk.

The usual price is $39.95. Our price **$33.89.** But until April 30, you can have it for **$27.99** (#9380).

MONEY TO BURN

If you've got money to burn and want the best monitor for Apple around and do a lot of word processing, you have no choice but The Genius. At only **$1695** (we'll ship free in U.S.) it includes an 80-column board, a special program to make it compatible with Word Star, and your choice of B/W, amber or green screen (only one).

What makes this monitor so special is that it displays a full 57 rows. This means you can see a full page displayed, just like it will be printed. This is unheard of for the Apple. (#113).

ULTIMA II SPECIAL

The hottest new game out is On-Line's Ultima II at $59.95. Here's a special you won't beat. Buy any item from us (no matter how small) and you can have the Ultima II for only **$37.99.** This special ends April 30, 1983.

Included in Ultima II is the fanciest packaging ever done by On-Line, a beautiful four-color 17x22 cloth map, suitable for framing, two disks (three sides) and hours of fun. (#1114)

NEW COMPUTER

We're in the process of installing a new Sage computer to run our business. This 16-bit, 68000 computer combined with Flexware software (also available for the Apple) will allow us to have the fastest system ever installed by any mail order business.

In less than the time it takes to type your name, we'll be able to tell you the status of your order, call up any invoice in the last year, tell you shipping cost for any hardware item, when back-ordered items will come in and much more.

We're quite excited about it and will be selling Sage and Flexware and doing custom programming for it. More on this later.

RANA DRIVES

We are now quite competitive on our Rana Drive prices. Give us a call for our new lower prices.

We also have the new Gibson high speed light pen at a discount. This is a knockout item. At press time our price was $296. Call for latest pricing. (#114)

Ice Demons is a nice new arcade game from the talented Matthew Jew. Listing at $29.95, our special this month is **$22.95.** (#7720).

Omega Microwave has an excellent new game out called A City Dies Whenever Night Falls. List price is $29.95 and our special is **$22.95.** The documentation that comes with this is unbelievable. And it is copyable and listable. (#707)

128K — $399

Also from Omega, we're happy to be carrying their Ramex-128 128K board for only **$399** (a bargain even at the full list price of $499). It requires no removing of chips to install. It comes with powerful disk emulation software, which adds eight new DOS commands. It is the only 128K board that allows the user to load or save a full 136K VisiCalc file in 20 seconds. (#708)

NEW

#7071 Pleasure (adults only) Village **$25.39**
#7070 Hands On! (adults only) Village **$25.39**
#9061 Prism (storybook and games) **$16.89**
#9681 Sheila - H.A.L. Labs **$21.19**
Rediform Redibinder - Great! ..... CALL
Flexware - The best, most flexible, and most expensive accounting software ever made for the Apple. ..... CALL
#8261 Lovers or Strangers **$25.39**
Corona IBM PC look alike. Dynamite! ..... CALL
#117 The Toaster (two removable 5 meg cartridges) by Xcomp **$2499.00**
#118 IDS paper feeder **$439.00**
#119 CP/M Card (TM) - CP/M 3.0 (TM) + 64K memory + 6MHz speed. ..... Call for price
AgDisk Agricultural software ..... CALL
#122 Compu-Music - Roland **$429.00**

#230 Transtar 130 daisy wheel printer with boldface and underscore. Six mo. ltd. warranty **$749.00**
#682 Taxan RGB color with board & cable ... **$399.00**
All Santa Clara and Davong drives available at discount
#7380 Money Decisions (Eagle) special **$149.00**

The Transtar 315 is a mind-blowing printer. It will dump any HIRES color screen in four-colors to the printer and to the paper. If you're playing a game, press the button and in a few seconds you'll have a four color printout of the screen. Traction or friction. Unbelievable. Should be ready for shipment shortly. Our price **$699.00** complete! (#231)

#33 PSIO Dual Function-Card - Videx **$189.00**
#240 SRW Color Coder-5 different color library cases for carrying floppies **$15.99**
We now have the complete line of Okidata printers. CALL

Child's Play is an incredible new piece of software written by Mike Taylor for children three to seven years old. It includes an etch-a-sketch, a series of mazes in which the cutest ant you ever saw is guided to his musical reward, and a series of quizzes which teach a child concepts of bigger than, different from, etc. Published by Huntington (with our daughter in mind) we have priced this so everyone can afford to enjoy it. The disk is crammed-packed and is only **$19.99.** Order #8999.

SSM has the hottest new modems in the business. Compatible with just about everything. The following specials are good through April.
#8562 Modemcard (300 baud) **$239.00**
#8563 Modem 1200 (1200 BAUD) **$549.00**
The following Transpaks include the ModemCard and the Source:
#8564 Transpak-1 (includes Transend 1) **$309.00**
#8565 Transpak-2 (includes Transend 2) **$349.00**
#8566 Transpak-3 (includes Transend 3) **$529.00**
The following include 1200 baud modem and the Source
#8567 Transpak-2+ (includes Transend 2) **$799.00**
#8568 Transpak-3+ (includes Transend 3) **$899.00**
The following include the Source:
#8560 Transend 1 ..... **$75.00**
#8561 Transend 2 ..... **$119.00**
#8569 Transend 3 ..... CALL

WE HAVE HUNDREDS OF ATARI PROGRAMS IN STOCK. GIVE US A CALL.

The absolutely most incredible program we carry is The Word Processor - the complete Bible on eight double sided disks plus one program disk. It will scan, search, and do unbelievable things. You'll never find a better bargain. Sale price **$149.99.** (#7320)

Call Toll-Free **800-344-5106** (outside California)

HUNTINGTON COMPUTING
Post Office Box 1297
Corcoran, California 93212

Foreign Orders 209-992-4481
In California 800-692-4146

Apple* is a registered trademark of Apple Computer, Inc.
Pet* is a registered trademark of Commodore.
TRS-80* is a registered trademark of Tandy Corp.
Atari* is a registered trademark of Atari, Inc.

Outside Calif. 800-344-5106

We take **MasterCard, American Express or VISA** (Include card # and expiration date). California residents add 6% tax. Include $2.00 for postage. Foreign and hardware extra. Foreign (excluding Canada): remit U.S. currency, checks on U.S. banks, use listed charge cards, or make direct wire transfers through Security Pacific Bank, Corcoran, for a $6.00 charge. All overseas orders shipped by air. Send for free catalog. Prices subject to change without notice.

MICRO™

Apple Slices

By Tim Osborn

This month's program, BUILDIT, demonstrates how programs external to VisiCalc can create and access VisiCalc worksheet files. Both VisiCalc novices and pros will learn from the following discussion.

VisiCalc uses three file formats for data storage: DIF (Data Interchange Format), standard worksheet files, and print format files. DIF is an excellent, well-documented communication aid that many packages use to send and receive information to and from VisiCalc. But, because DIF is designed for flexibility, it can be used independently of VisiCalc. To make DIF general and flexible, it was necessary to remove the formulas from the worksheet, storing only the results of these formulas instead. The print files just store an image of the worksheet and, like DIF, do not include formulas. To save formulas it is necessary to use the "/SS" command, which creates a standard worksheet file.

After some investigation I found that these worksheet files are no more than a VisiCalc EXEC file that contains the data in the worksheet just as you would type it in. For example, the following worksheet

	A	B
1	ALPHA	10
2	BETA	5
3	GAMMA	5

(where B3 = B1 – B2 (GAMMA = ALPHA – BETA)) would be stored as shown in figure 1.

With this method you could enter this worksheet into VisiCalc from the keyboard. The only unexpected thing is the last entry. The /X appears to be an undocumented VisiCalc command. The "/X–" tells VisiCalc to set the cursor direction to horizontal. The "/X>A1:" tells it to make A1 the upper left-hand corner of the screen. The "> A1:" sets the cursor at A1.

With this information I was able to write BUILDIT — a VisiCalc template building aid. BUILDIT builds accounting worksheets that itemize entries vertically; the categories they belong to are produced horizontally. BUILDIT prompts the user for the categories and iems. The relationship between the various items is described to BUILDIT by placing a relationship operator in front of each item as it is entered. BUILDIT accepts four types of operators:

1. A summable group member (a member in a list of items that produces a sum) signified by a "+" in the first character position. There must be at least two members of any summable group, each member must be preceded by a "+", and the list must be terminated with an "=" item entry (see below).
2. A stand-alone sum (an item to be entered as a lump sum rather than itemized) signified by a "/" in the first position of the item entry.
3. A sum, signified by an "=" in the first character position of the item. When BUILDIT encounters a sum following a summable group, it sets the template up to place the total of the preceding group in the row of this sum for all categories. If the sum is not immediately preceded by a summable group it sets the template up to sum the last two items entered that began with a "/", "=", or a "–" and stores the result in this row for all categories. If two items to sum were not previously entered, an error message will be produced and the entry will not be accepted.
4. A difference, signified by a "–" in the first position of the entry. A difference takes the last two items that began with a "/", "–", or an "=" and sets the template up to subtract the last item (highest numbered, lowest down in the worksheet) from the second to the last, storing the result in this row.

An Example

Let's say you want to set up a worksheet to handle sales (disks + books) less expenses (fixed + variable), compute the gross income, subtract taxes, and compute the net income. You also want to break down the worksheet by the first three months of the year (January through March) with a grand total column for the three months.

BUILDIT prompts you first to enter the categories in the category maintenance mode. You just enter each category (JAN., FEB., and MAR.,) one at a time. There is no need to enter a grand total category because BUILDIT always generates it for you. When you are done entering categories just enter "Q" for quit (you are prompted for this in case you forget), and BUILDIT will give you a chance to make any editorial changes in the category edit mode.

When you are done editing the categories just enter "Q" and BUILDIT will enter the item maintenance mode. BUILDIT then prompts you to enter

Figure 1

>B3:+B1–B2	Goto B3 and put the formula B1 – B2 there.
>A3:"GAMMA	Goto A3 and put the label "GAMMA" there.
>B2:5	Goto B2 and put the value 5 there.
>A2:"BETA	Goto A2 and put the label "BETA" there.
>B1:10	Goto A1 and put the value 10 there.
>A1:"ALPHA	Got A1 and put the label "ALPHA" there.
/W1	Set global parameter – one window.
/GOC	Set global order of recalculations to columns.
/GRA	Set recalculations on automatic.
/GC9	Set column width to 9 characters.
/X–/X A1: A1:	See below. See text.

item number 1. Item 1 would be DISKS, which is one source of SALES (a member of the summable group SALES). Inform BUILDIT of this by entering "+DISKS". Next you enter "+BOOKS" to tell the program that "BOOKS" is the second member of this group. Entering "=SALES" tells the program to set the worksheet to total the "DISK" and "BOOKS" entries and place the result in "SALES". The next three entries follow the same logic: 1. "+FIX. EX.", 2. "+VAR. EXP.", and 3. "=TTL EXP." (FIXED EXPENSE + VARIABLE EXPENSE = TOTAL EXPENSE).

The gross is the difference between SALES less TOTAL EXPENSES, so the next entry would be "-GROSS". The next item, taxes, is not itemized or computed so it is considered a stand-alone sum. Taxes would be entered as "/TAXES". The only thing left to do is compute the net income, which is GROSS - TAXES. The net income item would thus be entered as "-NET".

Now enter "Q" to end item maintenance and enter item edit mode where you can make any editorial changes. When you are done editing, enter "Q". You will then be prompted to enter a file name to save the worksheet/template. After the program has finished writing out your file, run VisiCalc and use the "/SL" command to load the worksheet/template. See the template in figure 2 (I have replaced the zeros with the formulas for the given worksheet coordinate). Notice that the operators are stripped off from the items and the grand total column is automatically generated.

BUILDIT is fully interactive and all relationships are validated up front. If they are not correct, you are told which operators would be valid. You are always prompted and never left wondering how to respond. It is simple to use (only four operators) but is designed for a limited number of applications. It is designed for accounting applications where only sums and differences are used. With modifications it could be made to handle all but the most complicated worksheet/templates. Perhaps as important as its application, is the fact that it demonstrates how programs external to VisiCalc can be used to create VisiCalc templates.

Figure 2

A	B	C	D	E
1	JAN.	FEB.	MAR.	GRAND TTL.
2 DISKS				@SUM(B2...D2)
3 BOOKS				@SUM(B3...D3)
4 SALES	@SUM(B2...B3)	@SUM(C2...C3)	@SUM(D2...D3)	@SUM(B4...D4)
5 FIX. EXP.				@SUM(B5...D5)
6 VAR. EXP.				@SUM(B6...D6)
7 TTL.EXP.	@SUM(B5...B6)	@SUM(C5...C6)	@SUM(D5...D6)	@SUM(B7...D7)
8 GROSS	+B4-B7	+C4-C7	+D4-D7	@SUM(B8...D8)
9 TAXES				@SUM(B9...D9)
10 NET	+B8-B9	+C8-C9	+D8-D9	@SUM(B10...D10)

Listing 1

```
10  GOSUB 1000: REM  INITIALIZE
15 ARY$ = "CATEGORY":PROCESS$ = "MAINTENANCE"
20  GOSUB 2000: REM  GET + EDIT CATEGORIES
21  FOR J = Ø TO NUM:CT$(J) = IN$(J): NEXT
22 NC = NUM: REM  SAVE NUMBER OF CATEGORIES
25 ARY$ = "ITEM":PROCESS$ = "MAINTENANCE"
30  GOSUB 2000: REM   GET + EDIT ITEMS
31  FOR J = Ø TO NUM:IT$(J) = IN$(J): NEXT
32 NI = NUM: REM  SAVE NUMBER OF ITEMS
50  GOSUB 4000: REM BUILD FILE
60  PRINT CD$"CLOSE ";FILE$
70  END
1000 CD$ =  CHR$ (4): HOME
1010  DIM IN$(6Ø): DIM CT$(6Ø): DIM IT$(6Ø)
1020  FOR K = Ø TO 8: READ MSG$(K): NEXT
1500  RETURN
2000 J = Ø:NODE = Ø
2005  GOSUB 2010: GOTO 2015
2010 LN = (4Ø - ( LEN (ARY$) +  LEN (PROCESS$))) / 2
2011  HOME : FOR X = 1 TO LN: PRINT " ";: NEXT :
      INVERSE : PRINT ARY$;" ";PROCESS$: NORMAL : RETURN
2015  IF J = 6Ø THEN  HTAB 9: VTAB 23:
      INVERSE : PRINT "MAXIMUM ";ARY$;" REACHED":
      FOR X = 1 TO 2000: NEXT X: NORMAL : GOTO 2028
2016  HTAB 2: VTAB 1Ø: PRINT "ENTER ";: INVERSE :
      PRINT "Q";: NORMAL : PRINT" TO END ";
      ARY$;" MAINTENANCE"
2017  HTAB 2: VTAB 12: PRINT ARY$;" NUMBER ";J + 1;:
      INPUT " NAME ";IN$(J)
2020  IF IN$(J) = "" THEN  GOSUB 2010: GOTO 2015: REM DONT ALLOW NULL
2021  IF IN$(J) = "Q" AND J = Ø THEN  INVERSE :
      HTAB 1: VTAB 22: PRINT "YOU MUST MAKE AT LEAST ONE ENTRY";:
      NORMAL : FOR K = Ø TO 2000: NEXT K: GOSUB 2010: GOTO2015
2022  IF IN$(J) = "Q" THEN IN$(J) = "": GOTO 2028
2025  IF ARY$ = "ITEM" THEN  GOSUB 7000:
      REM CHECK FOR "+" "-" "/" OR "="
2027 J = J + 1: HTAB 12: VTAB 12: GOSUB 9000: GOTO 2005
2028  IF ARY$ = "CATEGORY" THEN  GOSUB 2010: GOTO 2034
2029  IF NODE = 3 OR NODE = 6 THEN  GOSUB 2010: GOTO 2034
```

Listing 1 *(Continued)*

```
2030  GOSUB 9100: GOTO 2017: REM DISPLAY ERROR MSG+CONT.
2034  HTAB 5: VTAB 1Ø: PRINT "EDIT ";ARY$;:
      INPUT " (Y)ES OR(N)O ";A$
2035 NUM = J: REM  SAVE NO. OF ENTRIES
2036  IF  LEFT$ (A$,1) = "Y" THEN  GOSUB 2500: GOTO 2040
2038  IF  LEFT$ (A$,1) <> "N" GOTO 2034
2040  RETURN
2500 NUM = J: REM SAVE NUMBER OF ENTRIES
2502 PROCESS$ = "EDITING"
2503  GOSUB 2010: VTAB 2: HTAB 1:SP$ = "  "
2505  FOR I = Ø TO NUM STEP 3
2506  IF I = 9 THEN SP$ = " "
2507  IF IN$(I) = "" THEN  GOTO 2535
2510  HTAB 1: PRINT I + 1;SP$; LEFT$ (IN$(I),9);
2515  IF IN$(I + 1) = "" THEN  GOTO 2535
2520  HTAB 14: PRINT I + 2;SP$; LEFT$ (IN$(I + 1),9);
2525  IF IN$(I + 2) = "" THEN  GOTO 2535
2530  HTAB 28: PRINT I + 3;SP$; LEFT$ (IN$(I + 2),9)
2535  NEXT I
2540  VTAB 23: HTAB 1: PRINT "CHANGE NO.? ,ENTER ";:
      INVERSE : PRINT "Q";: NORMAL :
      PRINT " TO END ";: INPUT "";A$
2545  IF A$ = "Q" THEN  RETURN
2550 LN =  LEN (A$):GD$ = "Y":ZRO$ = "Y"
2560  FOR I = 1 TO LN:MD$ =  MID$ (A$,I,1)
2565  IF MD$ > "Ø" AND MD$ < ":"
      THEN ZRO$ = "N": GOTO 2575
2567  IF MD$ = "Ø" GOTO 2575
2570 GD$ = "N":I = LN: REM  REQUEST IS NOT NUMERIC
2575  NEXT
2576  IF GD$ = "N" THEN MSG$ = "NUMERIC": GOTO 2580
2577  IF ZRO$ = "Y" THEN MSG$ = "NON ZERO": GOTO 2580
2578  GOTO 2595
2580  VTAB 23: GOSUB 9000: VTAB 23: HTAB 1: PRINT "INPUT ";
      MSG$;" ";ARY$;" ";: INPUT "";A$: GOTO 2545
2595 NUM% =  VAL (A$): REM CONVERT TO NUMERIC
2600  IF NUM% > NUM THEN  VTAB 23: GOSUB 9000:
      VTAB 23: INPUT "REENTER, TOO HIGH ";A$: GOTO 2545
2602 IN$ = IN$(NUM% - 1)
```

(Continued)

Listing 1 *(Continued)*

```
2605  VTAB 23: GOSUB 9000: VTAB 23: HTAB 1:
      PRINT "CHANGE ";ARY$;" TO ";: INPUT "";IN$(NUM% - 1)
2610  IF ARY$ = "CATEGORY" THEN  GOTO 2503
2612 GD$ = "Y":NODE = Ø
2615  FOR J = Ø TO NUM - 1: GOSUB 7000:
      IF GD$ = "N" THEN  GOTO 2620
2616  NEXT
2617  IF NUM% < > NUM GOTO 2620: REM  SEE IF LAST ITEM CHANGED
2618  IF (NODE = 3 OR NODE = 6) THEN  GOTO 2620
2619 GD$ = "N": GOSUB 9100: FOR K = Ø TO 4000: NEXT K:
      REM DISPLAY ERROR MSG + DELAY
2620  IF GD$ = "N" THEN IN$(NUM% - 1) = IN$
2700  GOTO 2503
4000  DIM FRM$(NI,3):FIRST =  - 1:LAST =  - 1
4010  FOR K = Ø TO NI - 1:LT$ =  LEFT$ (IT$(K),1)
4015  IF LT$ = "+" AND LAST =  - 1 GOTO 4077
4020  IF LT$ = "+" THEN FIRST = LAST:LAST =  - 1: GOTO 4077
4060  IF LT$ = "=" THEN  GOSUB 4100
4070  IF LT$ = "-" THEN  GOSUB 4200
4072  IF LT$ = "/" THEN FIRST = LAST
4075 LAST = K
4077  NEXT : REM  K
4080  GOTO 4300
4100  IF LAST >  - 1 THEN  GOTO 4150
4105  FOR L = K - 1 TO Ø STEP  - 1
4110  IF  LEFT$ (IT$(L),1) < > "+" THEN  GOTO 4115
4112  NEXT
4115 FRM$(K,1) = "@SUM("
4120 FRM$(K,2) =  STR$ (L + 3) + "..."
4130 FRM$(K,3) =  STR$ (K + 1) + ")"
4140  RETURN
4150 FRM$(K,1) = "+":FRM$(K,2) =  STR$ (FIRST + 2)
     + "+":FRM$(K,3) = STR$(LAST + 2)
4170 FIRST = LAST: RETURN
4200 FRM$(K,1) = "+":FRM$(K,2) =  STR$ (FIRST + 2)
     + "-":FRM$(K,3) = STR$(LAST + 2)
4210 FIRST = LAST: RETURN
4300  HOME : VTAB 4: INPUT "ENTER FILE NAME ";FILE$: GOSUB 11000
4301  IF FILE$ = "" GOTO 4300
4302  ONERR  GOTO 4305
4303  PRINT CD$"DELETE ";FILE$
4304  PRINT CD$"OPEN ";FILE$: GOTO 4306
4305  PRINT CD$"OPEN ";FILE$: CALL 768: REM REPAIR ONERR DAMAGE
4306  FOR K = NI - 1 TO Ø STEP  - 1
4307 COL = NC + 2: GOSUB 4900:ROW$ =  STR$ (K + 2):
4308 A$ = ">" + COL$ + ROW$ + ":@SUM(B" + ROW$ + "..."
4309 COL = NC + 1: GOSUB 4900:A$ = A$ + COL$ + ROW$ + ")"
4310  GOSUB 5000: REM  WRITE RECORD
4315  FOR L = NC - 1 TO Ø STEP  - 1
4320 COL = L + 2: GOSUB 4900: REM  FIGURE LITERAL COLUMN NAME
4325  IF FRM$(K,1) = "" THEN 4340
4330 A$ = ">" + COL$ + ROW$ + ":" + FRM$(K,1)
      + COL$ + FRM$(K,2) + COL$ + FRM$(K,3)
4335  GOSUB 5000: REM  WRITE RECORD
4340  NEXT : REM  L
4345 A$ = ">A" + ROW$ + ":" +  CHR$ (34)
      + RIGHT$ (IT$(K), LEN (IT$(K))- 1)
4350  GOSUB 5000: REM  WRITE
4355  NEXT : REM  K
4360 COL = NC + 2: GOSUB 4900
4365 A$ = ">" + COL$ + "1:" +  CHR$ (34) + "GRAND TTL"
4370  GOSUB 5000
4375  FOR K = NC - 1 TO Ø STEP  - 1
4380 COL = K + 2: GOSUB 4900
4385 A$ = ">" + COL$ + "1:" +  CHR$ (34) + CT$(K)
4390  GOSUB 5000: NEXT
4400  RETURN
4900  IF COL > 52 THEN COL$ = "B" +  CHR$ (COL + 12): GOTO 4915
4905  IF COL > 26 THEN COL$ = "A" +  CHR$ (COL + 38): GOTO 4915
4910 COL$ =  CHR$ (COL + 64)
4915  RETURN
5000  PRINT CD$"WRITE ";FILE$
5005  PRINT A$: RETURN
7000 LFT$ =  LEFT$ (IN$(J),1)
7010  IF NODE = Ø GOTO 7100
7015  ON NODE GOTO 7200,7300,7400,7500,7600,7700,7800,7900
7100  IF LFT$ = "+" THEN NODE = 1: RETURN
7105  IF LFT$ = "/" THEN NODE = 3: RETURN
7110  GOTO 8000: REM  SEND ERROR MESSAGE
7200  IF LFT$ = "+" THEN NODE = 2: RETURN
7210  GOTO 8000
7300  IF LFT$ = "+" THEN  RETURN
7310  IF LFT$ = "=" THEN NODE = 3: RETURN
```

Listing 1 *(Continued)*

```
7315  GOTO 8000
7400  IF LFT$ = "+" THEN NODE = 4: RETURN
7410  IF LFT$ = "/" THEN NODE = 6: RETURN
7420  GOTO 8000
7500  IF LFT$ = "+" THEN NODE = 5: RETURN
7510  GOTO 8000
7600  IF LFT$ = "+" THEN  RETURN
7610  IF LFT$ = "=" THEN NODE = 6: RETURN
7615  GOTO 8000
7700  IF LFT$ = "+" THEN NODE = 7: RETURN
7705  IF LFT$ = "/" THEN  RETURN
7710  IF LFT$ = "=" THEN  RETURN
7715  IF LFT$ = "-" THEN  RETURN
7720  GOTO 8000
7800  IF LFT$ = "+" THEN NODE = 8: RETURN
7805  GOTO 8000
7900  IF LFT$ = "+" THEN  RETURN
7905  IF LFT$ = "=" THEN NODE = 6: RETURN
8000  HTAB 1: VTAB 22: GOSUB 9000
8005  HTAB 1: VTAB 22: INVERSE : PRINT "NUMBER ";
      J + 1;"'S 1ST CHAR. MUST BE ";MSG$(NODE): NORMAL
8010  FOR K = 1 TO 3500: NEXT K
8012 J = J - 1:GD$ = "N"
8015  RETURN
9000  PRINT "                                                  ";
9010  RETURN
9100  HTAB 1: VTAB 22: INVERSE : PRINT
      "1ST CHAR OF LAST ITEM MUST BE =,/ OR -";: NORMAL
9105  RETURN
10000  DATA  "+ OR /","+","+ OR =","+ OR /",
       "+","+ OR =","+, /, = OR ","+","+ OR ="
11000  FOR X = 768 TO 777: READ XX: POKE X,XX: NEXT
11002  RETURN
11005  DATA  104,168,104,166,223,154,72,152,72,96
```

MICRO™

ADVANCED

GRAF-PAK

Zoom HiRes Graphic Printing for Apple Computers

- Print front or back view of either or both screens
- Print upright, upside down, rotated left or right
- Selectable printing densities for many printers
- Easily place zoom viewport using on-screen crosshairs
- Large range of scale factors, independently selected
- Load files to either screen in just 5 keystrokes
- Type upper/lower case English or Greek text on screen
- Attach screen dump to your own programs, complete details
- Real Apple II DOS 3.3 format — Unprotected backup with COPYA
- Supports over 70 dot matrix and letter quality printers
- Supports serial, parallel, graphic, and buffer I/O cards
- Also works with the Basis and Franklin Computers
- Only $34.95 postpaid or see your dealer
- Versions without text annotation available for
  Apple II Pascal $34.95
  Apple III SOS 1.1 $44.95

2281 Cobble Stone Court
Dayton, Ohio 45431
513/426-3579

SmartWare

Dealer Inquiries Invited!

item number 1. Item
DISKS, which is one
member of the
SALES). Inform
entering "+
"+BOO
"BOO
this
t

lus, 48K

as
l

Belmont, CA

Description: The product provides the capability to create and manipulate large lists of information. List management is performed by high-speed search and sort routines.

Pluses: The system is expandable in that it will operate with from one to eight disk drives on line, providing management of up to 24000 records (4000 characters/record) of data. Built-in features permit printing form letters, mailing labels, and envelopes. A special feature for list backup is provided.

Minuses: You can use the program with only one disk drive, which makes it possible to destroy the program disk. The system should have been designed to prevent this rather than risk user forgetfulness. The company says only one list can be stored per disk because of the need to store large lists, but I think a multiple short list per disk option should have been built in. Neither of these problems are serious; the product is quite usable.

Documentation: Adequate. Numerous examples are provided instead of explanation.

Skill level required: The user needs exposure to the problems of list management to get maximum utility from this product.

Reviewer: Chris Williams

Product Name:	**ESTHER**
Equip. req'd:	64K TRS-80 Color Computer One disk drive, FLEX DOS
Price:	$54.95, $74.95 with source
Manufacturer:	Frank Hogg Laboratory 770 James St. Suite 215 Syracuse, NY 13203
Author:	Dale Puckett

Description: *ESTHER* shows how a computer is capable of artificial intelligence. *ESTHER* will remember your name and ask you questions in an effort to get you to unload some of your problems. If you want to show your friends what your computer can do, *ESTHER* will help break the ice. But tell *ESTHER* to "shut up!" and the program will end. It is written in assembly language and the responses are much faster than a similar program in BASIC. *ESTHER* accepts simple to complex sentences, but works best with shorter ones.

Pluses: *ESTHER* comes in several formats — the 6800, 6809, FLEX, or even Radio Shack version on disk. The program is fast and responds intelligently to simple sentences.

Minuses: Proper nouns must be capitalized for *ESTHER* to recognize them.

Documentation: A detailed manual includes clear instructions for loading and running; some of the major subroutines of the program are covered in detail. A little background on artificial intelligence programs is included.

Skill level required: None.

Reviewer: Bill Ball

Product Name:	**WP 6502 Version 1.3a**
Equip. req'd:	OSI Disk System
Price:	$250 (65D) $ 25 Upgrade from Version 1.3 Also available in 65U
Manufacturer:	Dwo Quong Fok Lok Sow 548 Broadway Suite 4F New York, NY 10012

Description: *WP 6502* is a full-feature word processor for OSI computers. Text files are created and edited with the TYPE, INSERT, DELETE, and REPLACE commands. Sentences and paragraphs can be rearranged *via* the block move utility. The global edit command allows all occurrences of "SMITH" to be replaced with "BROWN". Fixed segments of texts can be called into the current file by typing four control characters. All disk operations are performed by the file-clerk utility including LOAD, SAVE, RENAME, ERASE, and DIRECTORY.

Pluses: Owners of earlier versions of *WP 6502* can upgrade to revision 1.3a at a small additional cost. The file clerk utility includes a copy routine to initialize disks and make backup copies of any text files or even *WP 6502*. A memory test and disk test are included in the file clerk.

There is an INSTALL command, which allows *WP 6502* to be custom configured to your needs. Options include changing default parameters and assignments of control characters, as well as accommodating differences in terminal and printer character sets.

Minuses: Text files created by version 1.3 must be edited before running on version 1.3a to reflect the changes in the margin, tab, and line feed control characters. Editing is

done on the unformatted text file. In this mode, control characters are displayed but do not function and words may be split between lines. The user must flip back to the view mode to see the effect of his editing on the formatted output.

Documentation: The program is supplied with an operation manual and a training manual. The training manual is written for the non-computer user. The disk is supplied with a number of text files already in place complete with errors.

Skill level required: No computer knowledge necessary.

Reviewer: Earl D. Morris

Product Name: **Touch Typing Tutor**
Equip. req'd: VIC-20 (5K or more)
Price: $15.95
Manufacturer: Taylormade Software
8053 E. Avon Lane
Lincoln, NE 68505
Author: Marian Taylor

Description: The *Touch Typing Tutor* package contains two programs to teach the beginner how to type by touch rather than by "hunt 'n peck." The first program, LESSONS, is divided into 19 separate drills on the fingering of keys, ranging in difficulty from the "home row" in the first drill to punctuation in the last drill. The VIC tracks your progress and shows your % correct for each drill. The second program, PRACTICE, gives you random-letter sequences and then measures your speed and accuracy as you type in the sequences.

Pluses: Fun, useful, and well done! My kids (ages 9 and 10) love it; they learned where all the keys are effectively and enjoyably.

Minuses: Typing random letter sequences is not really a good test of typing speed. Random sentences would be more realistic.

Documentation: The 12-page manual is well written and quite comprehensive.

Skill level required: None.

Reviewer: David Malmberg

Product Name: **VIC Adventure Cartridges**
Equip. req'd: VIC-20 (5K or more)
Price: $39.95 each
Manufacturer: Commodore Business Machines, Inc.
487 Devon Park Drive
Wayne, PA 19087
Author: Scott Adams

Description: Commodore has released the first five of Scott Adams' classic Adventure games on cartridge for the VIC-20. These are outstanding games that allow you to indulge in fantasy role playing by giving your VIC simple one- or two-word commands, like GO NORTH, EXAMINE

What's eating your Apple®?

Find out with Apple-Cillin II™

If you use your Apple for your business or profession, you probably rely on it to save you time and money. You can't afford to guess whether it is working properly or not. Now you don't have to guess. Now you can find out with Apple-Cillin II.

Apple-Cillin II is the comprehensive diagnostic system developed by XPS to check the performance of your Apple II computer system. Apple-Cillin II contains 21 menu driven utilities including tests for RAM memory, ROM memory, Language Cards, Memory Cards, DISK system, Drive Speed, Keyboard, Printer, CPU, Peripherals, Tape Ports, Monitors and more. These tests will thoroughly test the operation of your Apple, and either identify a specific problem area or give your system a clean bill of health. You can even log the test results to your printer for a permanent record.

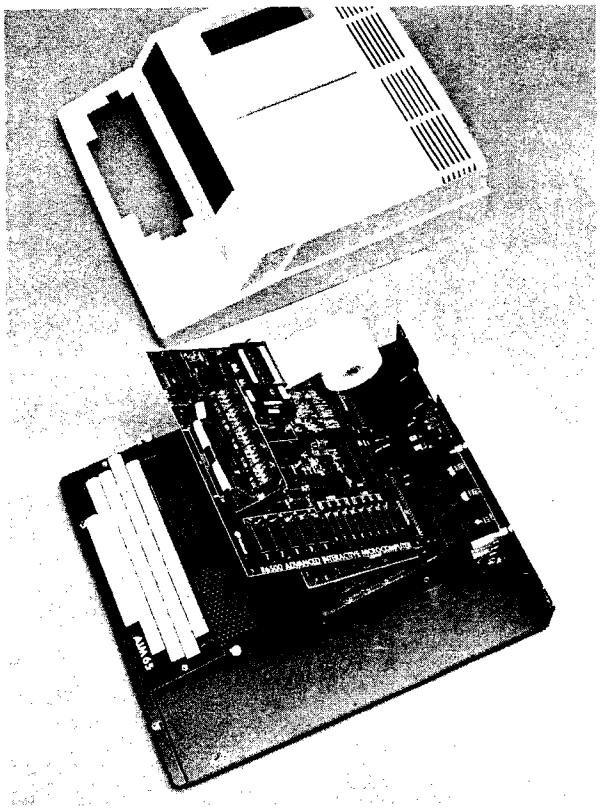
Apple-Cillin II works with any 48K Apple system equipped with one or more disk drives.

To order Apple-Cillin II - and to receive information about our other products - Call XPS Toll-Free: 1-800-233-7512. In Pennsylania: 1-717-243-5373.

Apple-Cillin II: $49.95. PA residents add 6% State Sales Tax.

XPS, Inc.
323 York Road
Carlisle, Pennsylvania 17013
800-233-7512
717-243-5373

Apple II is a registered trademark of Apple Computer Inc.



AIM HIGH

Let Unique Data Systems help you raise your sights on AIM 65 applications with our versatile family of AIM support products.

- Go for high quality with our ACE-100 Enclosure. It accommodates the AIM 65 perfectly, without modification, and features easy access two board add-on space, plus a 3" × 5" × 17" and a 4" × 5" × 15.5" area for power supplies and other components. $186.00.
- Get high capability with Unique Data System's add-on boards. The UDS-100 Series Memory-I/O boards add up to 16K bytes of RAM memory or up to 48K bytes ROM/PROM/EPROM to your Rockwell AIM 65. You also get 20 independently programmable parallel I/O lines with an additional user-dedicated 6522 VIA, two independent RS-232 channels with 16 switch-selectable baud rates (50 to 19.2K baud), and a large on-board prototyping area. Prices start at $259.00.
- If you need to protect against RAM data loss, the UDS-100B offers an on-board battery and charger/switchover circuit. $296.00.
- Heighten your AIM 65's communications range by adding the UDS-200 Modem board. It features full compatibility with Bell System 103 type modems and can be plugged directly into a home telephone jack via a permissive mode DAA. No need for a data jack or acoustic coupler. The UDS-200 also has software-selectable Autoanswer and Autodial capability with dial tone detector. The modem interfaces via the AIM 65 expansion bus, with the on-board UART and baud rate generator eliminating the need for an RS-232 channel. $278.00.
- The UDS-300 Wire Wrap board accepts all .300/.600/.900 IC sockets from 8 to 64 pins. Its features include an intermeshed power distribution system and dual 44-pin card edge connectors for bus and I/O signal connections. $45.00.
- Get high performance with the ACE-100-07 compact 4" × 5" × 1.7" switching power supply, delivering +5V @ 6A, +12V @ 1A, and +24V for the AIM printer. $118.00.

Installation kits and other related accessories are also available to implement your AIM expansion plans. Custom hardware design, programming, and assembled systems are also available. High quality, high capability, high performance, with high reliability . . . all from Unique Data Systems. Call or write for additional information.

Unique Data Systems Inc.
1600 Miraloma Avenue, Placentia, CA 92670

(714) 630-1430

Reviews in Brief *(continued)*

HOLE, JUMP, THROW AXE, etc. You can discover a pirate's treasures, find and disarm a bomb before it blows you up, or other daring exploits. These complicated and time-consuming games are loads of fun for adventurers of all ages. Games in progress may be saved on and loaded from tape. For added excitement, use the Votrax *Type 'N Talk* to run the first series of talking adventures.

Pluses: Great adventures creatively done! The cartridge format is easy to use, and allows a 16K program to run without any additional memory beyond the standard 5K. The talking feature is fun if you are fortunate enough to have a Votrax.

Minuses: These cartridges require you to LOOK (or "L" for short) at your surroundings whenever you move to a new location. This quickly becomes tedious. The documentation on the Votrax interface is wrong; it requires a 2400 baud rate rather than the 1200 rate claimed.

Documentation: Each of the five cartridges comes with the same well-written 12-page manual that explains the basics of adventuring, plus gives valuable hints for successful play.

Skill level required: None, other than imagination, perseverance, and luck.

Reviewer: David Malmberg

Product Name:	**Story Machine**
Equip. req'd:	Apple II with Applesoft or Apple II Plus with 48K RAM, DOS 3.3
Price:	$34.95
Manufacturer:	Spinnaker Software 215 First St. Cambridge, MA 02142 (617) 868-4700
Author:	Design Ware
Copy Protection:	Yes

Description: *Story Machine* is an educational program to help children, ages 5 to 9, write sentences, paragraphs, and simple stories. It develops vocabulary skills and keyboard familiarity. You type in simple sentences and the program acts them out in graphics.

Pluses: *Story Machine* is well written and interesting enough to keep a young child occupied for hours on end in writing short amusing stories. It is easy to use and has an informative guide. The graphics are illustrative and well done.

Minuses: The dictionary is limited and exacting. Perfect typing, not all that easy for a five-year-old, is required; no unnecessary spaces are allowed. The child must also be reading fairly well to run the program on his own.

Skill level required: Reading ability and interest in computers. Five years old might be too young.

Reviewer: Phil Daley

MICRO

Software Catalog

Name: **The DOS Enhancer (TDE)**
System: Apple II, DOS 3.3, ROM/RAM card
Memory: 48K
Language: Machine Language
Hardware: Disk II
Description: *The DOS Enhancer (TDE)* utility creates copyable DOS 3.3 disks that start up directly and quick-load a RAM card in 1.8 seconds. Its remarkable speed and efficiency comes from fixing all known "bugs," rewriting DOS (including the file manager) for enhanced speed of operation and assembling the resultant new *TDE* Quick DOS. Compatibility with standard Apple DOS 3.3 programs is maintained.
Price: $69.95
Includes utility disk, training/support disk, and complete documentation.
Author: Art Schumer
Available:
S&H Software
58 Van Orden Road
Harrington Park, NJ 07640
(201) 768-3144

Name: **Advanced X-Tended Editor**
System: Apple II, Apple II Plus; DOS 3.3
Memory: 48K
Language: Applesoft
Description: The *Advanced X-Tended Editor (AXE)* is a professional programming aid that provides the user with a text editor-style extension to the standard Applesoft operating system. *AXE* is called upon by one of over thirty commands that are easy, logical, and operate in the normal Applesoft entry mode or in *AXE's* editing modes. *AXE* operates on BASIC code as stored in memory by Applesoft. No conversion of code to text is required. In addition, all Apple II DOS and monitor commands are left fully functional.
Price: $69.95
Includes floppy diskette and complete documentation.
Available:
Versa Computing, Inc.
3541 Old Conejo Rd.
Suite 104
Newbury Park, CA 91320
(805) 498-1956

Name: **GraFORTH**
System: Apple II or Apple II Plus
Memory: 48K
Language: Machine Language
Hardware: DOS 3.3 and one or more disk drives
Description: *GraFORTH* is a fast compiled graphics language similar to FORTH, but with many built-in graphics features, including line and area graphics, Turtlegraphics, character graphics, and animated 3-D graphics.
Price: $75.00
Includes diskette and 220-page manual.
Author: Paul Lutus
Available:
Insoft, Inc.
10175 SW Barbur Blvd.
Suite 202B
Portland, OR 97219

Name: **Semi Draw**
System: TRS-80 Color Computer or TDP System 100
Memory: 32K
Language: Extended BASIC
Description: With *Semi Draw* your computer's keyboard or joystick draws in eight colors with semi-alpha graphics 8, 12, and 24. *Semi Draw* provides animation and dumps the picture of the screen to a Line Printer VII/VIII, NEC 8023, or C. Itoh 8510 printer. Just press the space bar to see the HELP display for instructions. This graphics program makes drawing with the Color Computer fun and easy for anyone six years and older!
Price: $21.95 cassette only
Includes cassette and instructions.
Author: Paul S. Hoffman
Available:
Computerware
P.O. Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **GraphPower**
System: Apple II, Apple III, IBM PC, DEC
Memory: 64K
Language: Pascal
Hardware: Disk drive
Description: *GraphPower* produces high-quality business graphics at low cost. Using data input from the keyboard or the Micro-DSS/Finance finanacial modeling system, *GraphPower* creates camera-ready graphics on paper or transparencies for presentations and produces graphs and charts including bar, stacked bar, side-by-side bar, line, pie, text, and reports. Features include automatic or manual scaling, eight letter sizes with five slants, up to four graphs per page, multiple image overlays, unlimited shading combinations, text annotation, mathematical operations, 260 data points, and more.
Price: $295.00
Available:
Ferox Microsystems, Inc.
1701 N. Ft. Meyer Dr.
Arlington, VA 22209
(703) 841-0800

Name: **Diversi-DOS™**
System: Apple II, Apple II Plus
Memory: 48K
Language: Assembly
Hardware: 16K - 128K RAM card optional
Description: *Diversi-DOS* is a new Apple DOS 3.3-compatible operating system that loads and saves BASIC, binary, and text files two-to-five times faster than standard DOS 3.3. *Diversi-DOS* also contains a keyboard type-ahead buffer and print-buffer utility. A simple, menu-driven, installation program is included on the unprotected disk.
Price: $30 by mail order only
Includes program disk and documentation.
Author: Bill Basham
Available:
Diversified Software Research, Inc.
5848 Crampton Ct.
Rockford, IL 61111

Name: **Stellar Shuttle**
System: Atari 400/800
Memory: 32K disk, 16K cassette
Language: Machine Language
Hardware: Joystick
Description: Assigned a dangerous rescue mission to the planet Ttam, you maneuver your space shuttle from the mother ship and begin a perilous descent to the surface, attempting to avoid streaking asterpods along the way. Retro rockets control your rate of descent to the narrow landing wells on the planet's surface where hostages eagerly wait for your arrival. The rockets also provide your only defense against pesky dragons that have a taste for Ttamians and will do their best to thwart your attempt to rescue the hostages and transport them safely to the mother ship. Four different levels of play are available.
Price: $24.95
Author: Matt Rutter
Available:
Broderbund Software, Inc.
1938 Fourth Street
San Rafael, CA 94901
(415) 456-6424

Name: **Veecee-Writer**
System: Apple II Plus, DOS 3.3
Memory: 48K
Language: Applesoft
Hardware: Disk drive
Description: *Veecee-Writer* translates VisiCalc (/PF) files for Apple Writer.
Price: $15.00
Includes instructions and copyable program disk.
Available:
Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

Name: **Ear Challenger**
System: Apple II Plus
Memory: 48K
Language: BASIC
Hardware: One disk drive
Description: Instructional lesson designed to teach elements of music to children or adults.
Price: $39.95
Includes documentation and diskette
Author: John M. Eddins and Robert L. Weiss, Jr.
Available:
Electronic Courseware Systems, Inc.
P.O. Box 2374, Station A
Champaign, IL 61820
(217) 359-7099

(Continued on next page)

Software Catalog *(continued)*

Name: **Hi-Res Plotting Package**
System: Apple II Plus with Applesoft ROM (DOS 3.2 or 3.3)
Memory: 48K
Language: Applesoft
Hardware: Disk drive, Epson printer with Graftrax (optional)
Description: *Hi-Res Plotting Package* features hi-res function plotting with a twist. Graphs are calculated and stored on disk, then viewed in rapid succession when the game paddle is turned. This package makes every Apple an oscillosope. A 3-D plotter (transparent and hidden line) is included along with many other useful math routines.
Price: $19.95
Includes floppy disk and complete instructions.
Author: William C. Jones
Available:
Apex Software Co.
8781 Troy St.
Spring Valley, CA 92077
(619) 466-2200

Name: **Colorcom/E**
System: TRS-80 Color Computer
Memory: 4K - 64K
Language: Machine Language
Hardware: ROMpak or diskette
Description: The *Colorcom/E* is a smart terminal program that comes in a ROM cartridge ready to plug in and run. Features and capabilities include on-line and off-line scrolling, off-line printing of data, receiving and sending cassette files, and support of any serial printer. Data can be easily edited before printing or writing to cassette or disk.
Price: $49.95
Includes manual.
Author: Mark Davidsaver
Available:
Spectrum Projects
93-1586 Drive
Woodhaven, NY 11421
(212) 441-2807

Name: **Crossword Scrambler**
System: Apple III
Memory: 128K
Language: Turn-key system
Hardware: Built-in disk drive and 80-character monitor
Description: *Crossword Scrambler* is an educational software product created to teach facts and spelling on five different subjects with graphically formatted screens and audio output. User-friendly prompts are designed for hands-on experience and computer interface. Data security concepts and password protection are demonstrated within the programs.
Price: $39.95 ppd.
(20% discount to bona fide educational institutions)
Includes diskette and documentation.
Author: David Cortopassi
Available:
SOFPROTEX
P.O. Box 271
Belmont, CA 94002

Name: **Micro Cookbook and Micro Barmate**
System: Apple II, Apple II Plus; DOS 3.3
Memory: 48K
Language: Compiled Applesoft BASIC and 6502 assembler
Description: *Micro Cookbook* and *Micro Barmate* are automated reference systems that instantly provide food or beverage recipes based on the ingredients the user has on hand. Drink and food recipes are selected *via* three methods: recipe name, category, and/or available ingredients. They also provide other food- and drink-related information — nutrition guides, calorie counter, party planning, etc. Both programs are fast and simple to use.
Price: $30.00
Includes software, basic recipe diskette, 28-page user manual, recipe index, and ingredient index.
Author: Joseph W. Butler III and Brian Skiba
Available:
Virtual Combinatics
P.O. Box 755
Rockport, MA 01966

Name: **Computer Slide Express**
System: Apple II Plus
Description: Apple Computer owners can convert computerized charts, designs, graphs, and graphics to 35 mm slides, prints, or overhead transparencies. With our new *Computer Slide Express*, Apple owners simply push a button to dial Visual Horizons in Rochester, transmit the information over ordinary telephone lines and receive by mail 35 mm color slides, standard size black-and-white prints, enlargements, or overhead transparencies.
Price: $6.00 each,
$30.00 minimum
Available:
Visual Horizons
180 Metro Park
Rochester, NY 14623
(716) 424-5300

Name: **Computer Football Strategy**
System: TRS-80 Models I and III, IBM PC
Memory: 32K - TRS-80
64K - IBM PC
Language: BASIC
Hardware: One disk drive
Description: Computer version of Avalon Hill's famous board game is based on the award-winning *Sports Illustrated* game of professional football. It forces the player to constantly make the right decisions about his team's offensive and defensive formations. Match wits against the computer or against a live opponent.
Price: $21.00
Includes diskette.
Available:
Avalon Hill Microcomputer Games
4517 Harford Road
Baltimore, MD 21214

MICRO™

IS THERE LIFE AFTER BASIC ?
YES ! WITH...
COLORFORTH™

MOVE UP FROM BASIC! Forth is a new, high level language available now for the TRS-80® Color Computer. **COLORFORTH**, a version of fig FORTH, has an execution time as much as 10 to 20 times faster than Basic, and can be programmed faster than Basic. **COLORFORTH** is highly modular which make testing and debugging much simpler. **COLORFORTH** has been specially customized for the color computer and requires only 16K. It does not require Extended Basic. When you purchase **COLORFORTH**, you receive **both** cassette and RS/DISK versions, the standard fig EDITOR and an extensive instruction manual. Both versions and 75 page manual $49.95

Add $2.00 shipping

Texas residents add 5 percent

DEALER AND AUTHOR INQUIRIES INVITED

ARMADILLO INT'L SOFTWARE
P. O. Box 7661
Austin, Texas 78712

MasterCard VISA

Phone (512) 459-7325

Hardware Catalog

Name: **Micromouse™**
System: Any
Description: The *Micromouse* is a small, hand-held device that can be interfaced easily to any microcomputer. When the mouse is moved on a table top, the cursor or pointer moves on the computer screen. The mouse has two butttons to draw lines on the screen. The buttons also can be used to identify, move, and position symbols.
Price: $180.00 in single quantities, $72.00 in quantities above a thousand. Includes instruction manual.
Available:
3G Company, Inc.
Rt. 3, Box 28A
Gaston, OR 97119
(503) 662-4492

Name: **Hypercartridge™**
System: Atari 400/800
Memory: 16K
Description: *Hypercartridge* gives hobbyists the ability to make their own cartridges at home. Software firms can market extensive ROM-based cartridges for use with 8K RAM (or more) computers without disk drives. It comes with four low-profile sockets for 24-pin ROMs or EPROMs (chips not included), two pin-select logic chips, and a capacitor. *Hypercartridge* can be used in two configurations: with any combination of 2532 EPROMs and 2332 ROMs; or with two Atari ROMs and two 2532 EPROMs or 2332 ROMs.
Price: $39.00/unit; quantity discounts available. Includes configured cartridge without EPROMs/ROMs
Available:
Chameleon™ Computing
Dept. of Physics and Astronomy
Box 119-P
Dickinson College
Carlisle, PA 17013
(717) 245-1717

Name: **Computer Practice Keyboard**
System: Any
Description: The printed keyboard is used to practice special function-key locations and to become familiar with all popular computers.
Price: $9.95 each
Includes shipping and handling.
Available:
Computer Practice Keyboard Company
616 9th St.
Union City, NJ 07087

Name: **Ink Stick**
Description: *Ink Stick* mounts inside most spool ribbon-dependent printers that use ½" spool ribbons and immediately replaces ink that is depleted from the ribbon to maintain an appropriate level of ink in the ribbon at all times. This extends the life of the ribbon frabic, reduces the operating cost of the printer, provides the user with consistent image density, and reduces the number of times ribbons are handled.
Price: $4.95 retail
Includes mounting cap, ½-oz. ink in container, applicator wick, and installation instructions.
Available:
Lawrence Electronics
3651 N. Cicero Avenue
Chicago, IL 60641

Name: **Sage II**
System: Sage II
Language: Pascal, FORTRAN, BASIC, Assembler
Description: *Sage II* offers the highest performance-per-dollar computer on the market in the price range of the IBM/Apple III, but with four to twenty times more computing power. Expansion capability includes more RAM, multi-user multi-tasking, hard drives, networks, and graphics.
Price: $4,450.00
Includes 128K RAM, 320K floppy, televideo 925 term, software
Available:
Sage Computer Technology
35 N. Edison Way, Suite 4
Reno, NV 89502

Name: **Data Vault™**
Description: *Data Vault* protects your computer tapes, disk cartridges, and floppy disks from the hazards of shipping and storage. They feature a rugged polyethylene exterior, internal shock-absorbing foam, and a positive-action locking system. Send for a catalog.
Available:
Kathy Sutherland
Sales Manager
Data Vault Division at PRC of America
475 Boulevard
Elmwood Park, NJ 07406
(201) 796-6600

Name: **ITALKII Speech Synthesizer**
System: Atari 400/800
Memory: 16K
Language: BASIC
Description: *ITALKII* offers an unlimited vocabulary, four voices, and powerful program development utilities. It requires no external power supply and outputs speech to the monitor's speaker. A machine-language driver allows *ITALKII* to speak while action graphics and sound effects are being executed. Software includes a dictionary, a word editor, a sentence builder, and *Word-blaster* (an arcade-style educational game).
Price: $199.00
Includes *ITALKII*, disk or cassette, manual, phonetic speech dictionary.
Available:
Greenbrier Marketing International, Inc.
509 South 48th St.
Suite 105
Tempe, AZ 85281
(602) 948-0005

Name: **Starfighter, The Ultimate Joystick**
System: Atari VCS, Atari 400/800, Commodore VIC, Sears Tele-Game
Description: This joystick controller is designed with an advanced mechanism that transfers movement directly from the user's hand through case-hardened steel components to the internal contacts. Its rounded shape helps to eliminate muscle fatigue when using other joysticks for long periods. Controller carries a two-year limited factory warranty.
Price: $16.95
Available:
SUNCOM, Inc.
270 Holbrook Dr.
Wheeling, IL 60090

Name: **Color III**
System: TRS-80 Color Computer
Memory: 4K - 64K
Description: If you are a do-it-yourselfer then you can upgrade a color computer to a 65-key keyboard with numeric pad, integrated television and computer display screen, telescopic antenna, internal disk and sound, channel selector knob, indicator lamps, ROM pack slot, and all I/O connector jacks mounted in a *Model III* enclosure. Color Computer products work as before.
Price: $15.00
Includes instruction manual, 12 professional drawings and templates, bill of material, check-off list, and manufacturing list.
Available:
L & E Electro Sales Co.
7017 Hazeltine Ave #10
Van Nuys, CA 91405
(213) 994-3110

Name: **Joyport**
System: Apple II or Apple II Plus
Description: *Joyport* expands the game port to use four fully functional Apple-compatible paddles and two Atari-type joysticks. No modification is necessary. The Joyport simply plugs into the existing game I/O port.
Price: $49.95
Includes user's manual.
Available:
Sirius Software
10364 Rockingham Drive
Sacramento, CA 95827

MICRO

MICRO™

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

101. 80 Micro (October, 1982)

Miller, Franklyn D., "The Colorful Computer — Part III," pg. 254-260.
A number of program listings for those without Extended Color BASIC in their TRS-80 Color Computer.

Degler, Roger L., "LP VII Patch for the CC," pg. 304-306.
An eight-bit printer-driver for the 6809-based Color Computer.

102. Popular Electronics 20, No. 10 (October, 1982)

Anon., "6809 FORTRAN," pg. 42.
Running under FLEX and UniFLEX, this compiler complies with ANSI FORTRAN-77 subset of FORTRAN.

103. Call —A.P.P.L.E. 5, No. 9 (September, 1982)

Anon., "Enhancement to The Mill," pg. 75.
MSM is an enhancement to The Mill Assembler Development Kit that combines the features of ASM09, ASM09IO.BIN and LOAD09. Thus, a stand-alone MSM09 BRUNable 6809 assembler.

104. Personal Computer World 5, No. 9 (September, 1982)

Anon., "Dragon 32," pg. 40-41.
The Dragon 32 is a 6809-based personal computer, 32K RAM, nine colors, high-resolution graphics, etc.

105. Commodore Microcomputer Magazine 3, No. 4 (August/September, 1982)

Kutz, Walt, "SuperPET Update," pg. 14.
COBOL for the 6809-based SuperPET, using extra memory, single board upgrade (8032 to SuperPET), accessing the serial port, etc.

Staff, "Commodore News," pg. 21.
Everything you always wanted to know about the 6809-based SuperPET — and asked! Questions and answers.

106. The Rainbow 2, No. 3 (September, 1982)

Lester, Lane P., "An Electronic Gradebook Can Make '82-'83 Much Easier," pg. 8-14.
A program for the TRS-80 Color Computer-equipped teacher.

Walrath, Del, "Let's Learn How To Do a Number-Picking Game," pg. 18-20.
A programming tutorial using a number-memory game.

Nolan, Bill, "Demons in the Dungeon? Let's See 'Em All," pg. 22-32.
A program to get quick information about a particular kind of demon without looking it up. A TRS-80 Color Computer dungeon game utility.

Harpe, David, "PUT, GET and Random Forms Make for Unusual Graphics," pg. 32-33.
A graphics program for the 6809-based Color Computer.

Blyn, Steve, "Make the Difficulty Level Variable," pg. 35-36.
Suggestions and an illustrative listing for CAI programs on the 6809-based TRS-80 Color Computer.

Hryzak, Wolfgang, "Ping-Pong International Game for People and 80C," pg. 41-43.
A German game for the 6809-basd Color Computer.

Roslund, Charles J., "An Automatic Key Repeat Feature is Handy to Have," pg. 47-49.
Add auto-key repeat to your Color Computer with this machine-language routine.

Schmidt, Jim, "Here Are Some Useful Utilities for Your Use," pg. 51-53.
Utilities for the Color Computer include a line-width driver, an 8-bit graphics driver, and a speed routine.

Rosen, Bob, "The Simple Way to 64K," pg. 59.
A hardware modification to convert the 6809-based Color Computer to 64K.

Hine, Al, "The Track Will Provide Hours of Varied Fun," pg. 60-70.
A race game for the 6809-based Color Computer.

107. 80-U.S. Journal 5, No. 10 (October, 1982)

Beste, Steve Den, "Word Processing on Your Color Computer," pg. 65-74.
A line-oriented text editor for the TRS-80 Color Computer with disk.

Fawcett, Dale H., "Serial Printer Interfacing," pg. 95-97.
Connecting to your 6809-based Color Computer.

108. MICRO No. 58 (October, 1982)

Whiteside, Tom, "Apple Pascal P-Code Interpreter and the 6809," pg. 79-84.
A rewrite of the Apple UCSD Pascal P-Code interpreter for the MC6809 shows code size and speed improvements.

Tenny, Ralph, "A Homespun 32K Color Computer," pg. 91-95.
A hardware article detailing a relatively simple memory expansion from 16K to 32K in the Color Computer.

109. MicroComputer Printout 3, No. 11 (October, 1982)

Preston, Chris, "MicroScope," pg. 20-21.
Multiple processors and how they work including 6502/Z80, 6502/6809, and similar combinations.

110. Compute! 4, No. 10 (October, 1982)

Anon., "TRS-80 Color Computer Program," pg. 223.
Colortext for the 6809-based TRS-80 Color Computer is a high-resolution text driver that displays a variety of character fonts and graphics on the screen simultaneously.

111. Personal Computer 5, No. 10 (October, 1982)

Oliver, Roger and Sadler, Chris, "Positron 9000," pg. 128-134.
The Positron is a new British-made microcomputer using 6809 with 64K RAM.

112. '68 Micro Journal 4, Issue 10 (October, 1982)

Ney, Robert L., "Color User Notes," pg. 11-13.
Discussion of F-MATE Version 2.0, powerful business systems for the TRS-80 Color Computer, etc.

Commo, Norm, " 'C' User Notes," pg. 14-19.
Discussion of Intersoft Version 1.0, a small C compiler for the 6809.

Watson, Ernest Steve and Brady, F. Dale, "Home Accounting Program," pg. 20-23.
Part III of an accounting system for 6809 systems.

113. The Rainbow 2, No. 4 (October, 1982)

Ridge, Herbert B., "Pope Gregory Would Like This Calendar," pg. 8-12.
A calendar program for the TRS-80 Color Computer.

Inman, Don, "Let's Learn How to Use Graphics with CoCo," pg. 14-17.
A graphics tutorial for the 6809-based Color Computer.

Garrett, Ron, "Key Checks to Various Accounts with This Program," pg. 20-24.
A Color Computer program to allow you to designate spending areas for a check or payment into different categories for business expenses.

MICRO™

Language Packages

Editor's Note: This list of Language Packages is not meant to be comprehensive.

Ada				
68000	Ada Compiler	TeleSoft	q-Bus, Multi-bus, S-100 configurations	$250
Z80, 8080, 8086, 8088	Ada	Supersoft	CP/M, MSDOS, 32K RAM min.	$350
APL				
Ap II	APL	Vanguard	Z80 softcard, RAMcard, 1 drive, 48K	$500
BABBLE				
Ap II	BABBLE	Software Factory	16K cass/32K disk	$15 (cass) $20 (disk)
CEEMAC				
Ap II/II	CEEMAC	Vagabondo	48K	$75
COBOL				
Ap II/II Plus	COBOL-80	Microsoft Enterprises	2 drives/44K/Softcard	$750
OSI	COBOL	Prism Software		
6809, 68000	RM/COBOL	Ryan-McFarland Data Compass Corp.	Unix(68000), 8 disk, 1 drive, 64K (6809)	$750 - $1250
OSI C-3	COBOL 3.3	OSI	CP/M	$600
Ap II/II Plus	Nevada COBOL	Ellis Computing	Softcard, CP/M, 1 drive	$29.95
6502	CIS COBOL	MicroFocus	UNIX or CP/M	$850
Ap II/II	Nevada COBOL	FORTH, Inc.	Softcard, CP/M, 1 drive	$29.95
COMAL-80				
PET	COMAL-80 (best of BASIC & Pascal)	Metanic	CP/M	
FOCAL-65				
Ap II, SYM, AIM, OSI, KIM	Fast FOCAL-65	6502 Program Exchange	48K, 1 drive (Apple), 16K, cassette (others)	
FOLIO				
KIM-1	FOLIO	Soft Corp		$50 cass $3 extra
FORTE				
Ap II w/Integer	FORTE	SofTape		$29.95
FORTH				
Ap II/II Plus	TransFORTH II	Insoft, Inc.	1 or more disk drives/48K/DOS 3.3	$125
At	fig-FORTH	Atari	10K	
Ap II/II Plus	FORTH (Microspeed, superset of FORTH)	Applied Analytics, Inc.	4K RAM	$495 (2 MHz) -u Speed II $645 (4 MHz) -u Speed II Plus
Ap	SuperFORTH	Hayden Software	48K/DOS 3.2	$49.95 disk
TRS-80C	CC FORTH	Frank Hogg	Disk drive	$99.95
Ap II/II	FORTH 72 Version 2	MicroMotion	1 disk drive	$89.95/ $139.95 hi-res & floating point
6809	FORTH 6809	Talbot Microsystems		$100
SYM-1	SYM-FORTH 1,0	Saturn		$135
6502	fullFORTH	CGRS Microtech		$65
Ap II/II	FORTH-79	MicroMotion	1-6 disk drives, 48K, 80 column bd. optional	$99.95/ $139.95 with floating point & hi-res
OSI	Stand-Alone fig-FORTH	FORTH Tools		$49.95
68000	PolyFORTH			$5100-8200
6809	X-FORTH	Frank Hogg	FLEX, disk drive	$129.95
COM-64	C64 FORTH	Performance Micro Products		$79.95 cass/ $99.95 disk
VIC-20	VIC FORTH	HES		$59.95
Ap II/II Plus	FORTH 1.7	Information Unlimited	1 disk drive, 48K, Ap DOS 3.2	$140
Ap	FORTH I, II	SofTape	1 drive	$49.95
OSI-C3, 4, OS65D-3	S-FORTH	Aurora	20K	$34.95 (OS65 D-3), $49.95 with source listing
Ap II/II	GraFORTH	Insoft, Inc.	48K, 16K RAM card recommended, DOS 3.3, 1 drive	$75
PET	FORTH	AB Computers	16K or 32K, PET/CBM disk drive	$50
Ap II/II	Timin FORTH 3.5	Lifeboat Assoc.	Z-80 Softcard, CP/M or CDOS, 32K min., 2 drives	$250
TRS-80C	Color-FORTH	Armadillo Int'l. Software	16K	$49.95
KIM	FORTH	Eric C. Rehnke		$90 cass/$75 source code only
TRS-80C	Color-FORTH	MicroWorks, Inc.	4K	$109.95

Language Packages will be continued in next month's issue.

Language Packages

Addresses

AB Computers
252 Bethlehem Pike
Colmar, PA 18915
215-822-7727

Abacus Software
PO Box 7211
Grand Rapids, MI 49510
616-241-5510

Addison Wesley Publishing Co.
Jacobs Way
Reading, MA 01867
617-944-3700

Apple Computer, Inc.
Software Division
10260 Bandley Drive
Cupertino, CA 95014
516-751-5139

Applied Analytics, Inc.
8235 Penn Randall Place, Suite 202
Upper Marlboro, MD 20972
301-420-0700

Armadillo International Software
PO Box 7661
Austin, TX 78712
512-459-7325

ATARI Home Computers
1265 Borregas Ave.
Sunnyvale, CA 94086
800-538-8543
408-745-2100

Aurora Software Assoc.
37S. Mitchell
Arlington, IL 6005
312-259-3150

CGRS Microtech
PO Box 102
Langhorne, PA 19047
215-757-0284

Commodore Business Machines, Inc.
681 Moore Rd.
King of Prussia, PA 19406
215-6987-9750

Compu/Think
965 West Maude Avenue
Sunnyvale, CA 94025
408-245-4033

Computerware
PO Box 668
Encinitas, CA 92024
619-436-3512

Creative Solutions
4801 Randolph Rd.
Rockville, MD 20852
301-9844-0262

Datasoft, Inc.
19519 Business Center Drive
Northridge, CA 91324
213-701-5161

Duggers Growing Systems
POB 305
Solano Beach, CA 92075

Dynasoft Systems, LTD
P.O. Box 51
Windsor Junction
Nova Scotia, CANADA
B0N 2V0
902-861-2202

Ellis Computing
3917 Noriega Ave,
San Francisco, CA 94122
415-753-0186

Eric C. Rehnke Technical Services
1067 Jadestone Lane
Corona, CA 91720
714-371-4548

FORTH, Inc.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
213--372-8493

FORTH Interest Group
POB 1105
San Carlos, CA 94070
415-962-8653

Frank Hogg Laboratory
130 Midown Plaza
Syracuse, NY 13210
315-474-7856

Gnosis
4005 Chestnut St.
Philadelphia, PA 19104
215-387-1500

Hoyt Stearns Electronics
413E Cannon Dr.
Phoenix, AZ 85036

Human Engineered Software
71 Park Lane
Brisbane, CA 94005
415-468-4116

Information Unlimited Software, Inc.
281 Arlington Ave.
Berkeley, CA 94707
415-525-9452

Insoft, Inc.
10175 S.W. Barbur Blvd.
Suite 202B
Portland, OR 97219
503-244-4181

JRT Systems
1891 23rd. Avenue
San Francisco, CA 94122
415-566-5100

Kenyon Microsystems
3350 Walnut Bend
Houston, TX 77042
713-978-6933

Keyser Enterprises
22 Clover Lane
Mason City, IA 50401

Krell Software
21 Millbrook Dr.
Stony Brook, NY 11790
516-751-5139

Lifeboat Associates
1651 Third Ave.
New York City,NY 10028
212-860-0300

Lucidata
POB 128
Cambridge CB2 5EZ
England

Merrimack Systems
PO Box 5218
Redwood City, CA 94063

Metanic ApS
Kongevejen 177
DK-2830
Virium, Denmark

Micro Focus, Inc.
1601 Civic Center Drive
Santa Clara, CA 95050
408-496-0176

Micro Motion
12077 Wilshire Blvd. 506
Los Angeles, CA 90025
213-821-4340

Micronetics Design Corp.
932 Hungerford Dr., Bldg. 11
Rockville, MD 20850
301-424-4870

Microsoft
400-108th Ave. NE
Bellevue, WA 98004
206-828-8080

MicroWorks
PO Box 1110
Del Mar, CA 92014
619-942-2400

Muse Software
347 N. Charles Street
Baltimore, MD 21201
301-659-7212

Omega Software
POB 70265
Sunnyvale, CA 94086
408-733-6979

On-Going Ideas
RD1, Box 810
Starksboro, VT 05487
312-259-3150

On-Line
37575 Mudge Ranch Rd.
Coarsegold, CA 93614

Ohio Scientific Instruments
7 Oak Park
Bedford, MA 01730
617-275-4440

Prism Software
PO Box 928
College Park, MD 20740

Quality Software
6660 Reseda Blvd., Suite 105
Reseda, CA 91335
213-344-6599

Querty Computer Systems
20 Worcester Rd.
Newton Hall, Durham
England LI.25 67045

Radcliffe House
66-68 Hagley Rd.
Edgbaston, Birmingham
United Kingdom B16 8PF

Radio Shack Educational Software Division
400 Atrium,
One Tandy Center
Fort Worth, TX 76102
817-390-3302

Ryan-McFarland Data Compass Corp.
3233 Valencia Ave.
Aptos, CA 95003
408-662-2522

Saturn Software Limited
8246 116A Street
Delta, British Columbia
CANADA V4C 5Y9

6502 Program Exchange
2920 W. Moana
Reno, NV 89509
702-825-8413

Softcorp
1372 East 52nd. St.
Chicago, IL 60615

SofTape Software Exchange
10432 Burbank Blvd.
North Hollywood, CA 91801
213-885-5763

SofTech Microsystems, Inc.
16885 West Bernardo Dr.
San Diego, CA 92127
714-942-1727

Software Factory
PO Box 904
Chatsworth, CA 91311

Sorcim
1333 Lawrence Expess-way, Suite 148
Santa Clara, CA 95051
408-727-7634

Succinct Systems
1346 River St.
Santa Cruz, CA 95060
408-426-4197

SuperSoft
PO Box 1628
Champaign, IL 61820
217-359-2112

Tallgrass Technologies Corp.
9009 W. 95th St.
Overland Park, KS 66212
913-381-5588

Tamarack
Water St.
Darby, MT 59829
406-821-4596

Telesoft
10639 Roselle St.
San Diego, CA
714-457-2700

Terrapin, Inc.
678 Mass. Ave., 205
Cambridge, MA 02139
617-492-8816

Vagabondo Enterprises
1300E. Algonquin-35
Shawnburg, IL 60195
312-397-8705

valFORTH International
3801 E. 34th St.
Tucson, AZ 85713
800-528-7070

Vanguard Systems, Corp.
6901 Blanko
San Antonio, TX 78216
512-340-1978

Volition Systems
POB 1236
Del Mar, CA 92014
619-481-2286

Williamsville Publishing Co.
Box 250
Fredonia, NY 14063

Wordsworth
PO 28954
Dallas, TX 75228
214-783-0419

NIBBLE *EXPRESS* Vol. 1

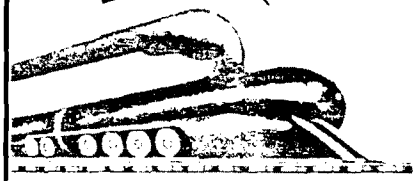
TABLE OF CONTENTS

	Page
EDITORIAL	3
APPLE TRAC — Personal Finance Management *by Mike H*	7
SORT 'EM OUT — Principles of Sorting *by NIBBLE Staff*	17
PSEUDO UTO-START — Reset with CTRL Y *by Rick Con*	17
INITIALIZE NEW FILES WITH ONERR GOTO *by NIBBLE*	18
MACHINE LANGUAGE SCREEN DUMP *by R.M. Mottola*	18
FREE? DISK SECTORS *by Chuck Hartley*	19
HI-RES SPACE MAZE — Graphics Game *by NIBBLE Staf*	22
UN-GRAPHIC GRAPHIC PRINTING *by NIBBLE Staff* ...	23
TABLE PRINTING MADE SIMPLE! *by NIBBLE Staff*	24
DYNAMIC ARRAY DIMENSIONING *by NIBBLE Staff*	26
BLOCKING VERY LARGE FILES *by NIBBLE Staff*	26
LOW RESOLUTION SHAPEWRITER — High Speed Actio	31
SPACE ANIMATION — Add ZIP to your Games *by NIBBL*	35
STAR ATTACK — Fast Hi-Res Conflict Game *by Mike Hai*	37
PADDLE READING IN ASSEMBLY LANGUAGE *by NIBBL*	38
FIRING HI-RES MISSILES — Aiming and Control *by NIBE*	39
AIRSHIP BATTLE — Join the Navy/Air Force Maneuvers *b*	47
AUTOMATIC TEXT ACCENTING *by NIBBLE Staff*	50
WATCH OUT FOR GRAPHICS OVERFLOW *by Mike Harv*	50
T.O.U.C.H. — Text Processing System *by Mike Harvey* ...	53
TOUCH — Large Hi-Res Characters *by William Reynolds III*	58
TOUCH FILE — Replace Option *by William Reyno*	59
DOUBLE/TRIPLE/AND MORE OVERPRINTING *by NIBBL*	59
ARROWS AND CONTROL CODES *by NIBBLE Staff*	59
APPLE TRICKS — Fast DOS/Spcl Chars/Unlistables *by C*	60
APPLESOFT VS. INTEGER BASIC PERFORMANCE *by A*	65
APPLE II — Paper Tiger Graphics *by Mike Harvey*	69
APPLESOMN — Fun Music and Fun *by Mike Harvey*	73
APPLE DISPLAY FILE CONTROL *by Alexander Laird*	77
APPLESOFT REM REMOVER *by Alan D. Floeter*	78
FAST FORECASTING AND SALES TRENDING *by Mike H*	83
SUPER WEAVER — Hi-Res Weaving Design *by Alexande*	89
FOOTBALL — Hi-Res Grid-Iron Action *by Lou Haehn* ...	93
BUILD DUAL JOYSTICKS FOR UNDER $15.00 *by NIBBL*	97
BUILD THE TWO TAPE CONTROL UNIT *by NIBBLE Staf*	98
DISK ZAP — THE EFFECTIVE ATTACK *by NIBBLE*	99
APPLE TRICKS — CORRECT Execution LIST *by Cra*	101
PIP I — Personal Inventory Program on Tape *by Rick Con*	103
FUN WITH ASSEMBLER — The Bell & Whistle *by Craig Cr*	106
PIP II — Plus Disc *by Rick Cornow*	113
PASSING VARIABLES IN APPLESOFT BASIC *by R.M. M*	119
MANAGING AND MOVING DISK BUFFERS *by William Re*	120
MONITOR EXECUTION — Basically *by William Reynolds*	121
AMPER-INTERPRETER — Add Print-Using and Much Mo	123
FUN WITH ASSEMBLER — Graphics *by Alexander Laird*	133
STRING FUNCTION FOR INTEGER BASIC *by William Re*	135
BASIC/MACHINE LANGUAGE SUBROUTINE CREATOR	135
CHR$ FUNCTION FOR INTEGER BASIC *by William Reyn*	135
FUN WITH ASSEMBLER — Alpha/Beeper *by Craig Cross*	136
APPLE A.I.M. — Automated Intelligent Mailing *by Michael*	139
APPLE CONCORDANCE — Track Variable and Line #'s *b*	147
LOW SCORE II — Strategy Game *by Rudy A. Guy*	153
HOW TO WRITE GAMES THAT LAST *by Mike Harvey* ...	157
IMPROVING THE MULTIPLE ARRAY SORT *by Rick Conr*	158
APPLE UPPER/LOWER CASE PRINTING *by Mike Harvey*	159
WILL O' THE WISP — High Adventure *by Mark Capella*	161
NIFFUM — DOS 3.3 to 3.2 Conversion *by C.J. Thompson*	169
BLAST AWAY! — Lo-Res Shooting Gallery *by Andrew Be*	171
FUN WITH MONITOR — How to Enter Assembly Langua	174

SECOND PRINTING!

BRAND NEW PROGRAM LISTINGS!

nibble EXPRESS! (for your Apple*)



ORDER NOW

All programs and Articles are centered on the Apple Computer family.

NIBBLE
P.O. Box 325
Lincoln, MA 01773

Yes! I want NIBBLE EXPRESS Vol. 1 in my library! Here's my ☐ Check ☐ Money order for $12.95 plus $1.75 postage/handling. (Outside U.S. add $2.75 Surface Mail or $5.00 Airmail.)

☐ Also send me NIBBLE EXPRESS Vol. 2 at $14.95 plus $1.75 postage/handling. (Outside U.S. add $2.75 Surface Mail or $5.00 Airmail.)

Master Card & Visa Accepted

Card # _____ Expires _____
PLEASE PRINT CLEARLY

Signature _____

Telephone _____

Name _____

Street _____

City _____ State _____ Zip _____

Your check or money order must accompany your order to qualify.

Outside U.S. Checks must be drawn on a U.S. Bank.

*Apple is a registered trademark of Apple Computer Company.

If you're looking for quality software at an outrageously reasonable price, we have a message for you...

Apple™ owners:

```
10 HOME: P$ = "GZPV UREV WLOOZIH LUU HLUGHRWV":VTAB12:
HTAB6: FOR P = 1 TO LEN(P$):J = ASC(MID$(P$,P,1)):PRINT
CHR$(ABS((155*(J >64)) – J));: NEXT:PRINT
```

TRS-80® owners:

```
10 CLS: DEFINT A-Z: PRINT@529,;: P$ = "GZPV UREV WLOOZIH
LUU HLUGHRWV": FOR P = 1 TO LEN(P$):J = ASC(MID$(P$,P,1)):
PRINT CHR$(ABS((155*(J > 64)) + J));: NEXT:PRINT
```

ATARI® owners:

```
10 GRAPHICS 0:DIM P$(30):P$ = "GZPV < UREV <
WLOOZIH < LUU < HLUGHRWV":FOR P = 1 TO 30:
?CHR$(155-ASC(P$(P)));: NEXT P:?
```

Type This Code!

Take a close look at the message above. If you own an APPLE™, an ATARI®, or a TRS-80® microcomputer, you probably recognize the language.

It's in BASIC...a one-line program you can type into your computer to find out how much money you can save on a one-year subscription to a unique, useful, entertaining software magazine...**SoftSide.**

Sure, we know this is a rather unconventional way to introduce you to **SoftSide**...but once you know what our magazine is all about, we know you'll agree that it's a rather unconventional publication.

You see, each month, between the covers of **SoftSide** we publish some of the most exciting games, practical utilities, and captivating adventures you'll find anywhere...at any price!

SoftSide publishes original programs, written by some of the most important names in the software business. Over half of each issue is devoted to BASIC line listings for programs you can type into your computer and enjoy forever. Programs that might cost you hundreds of dollars if you purchased them individually at your computer store, but actually cost you just a few cents each because of **SoftSide's** low, one-year subscription price of just $24.

And right now you can pay even less than $24 for the next 12 issues of SoftSide Magazine!

It's easy! Just type the one-line of code we've prepared for your system (above) into your APPLE™, ATARI® or TRS-80® microcomputer. Then type "RUN," and the unscrambled message on your monitor will tell you how much you can deduct from **SoftSide's** already low subscription price of $24.

Fill in the coupon below and send it along with your payment and we'll send you the next 12 issues of **SoftSide** Magazine!

SoftSide **Coded Bonus Offer**

Your Order	*Reg. Price	DEDUCT YOUR CODED BONUS	YOU PAY ONLY (Enter Amount)
☐ 12 Issues Of **SoftSide** Magazine	$24/yr.		

Send me the ☐ APPLE™, ☐ ATARI®, ☐ TRS-80® Version
(Please check one)

If you do not use the coded offer, simply pay this amount. You still save 1/3 off the newsstand price.

☐ **Yes, SoftSide** is the one computer software magazine I can't afford to be without.

Name _____

Address _____

City/State _____ Zip _____

Send to:

SoftSide Publications, Inc.
Dept M65
6 South Street
Milford, NH 03055

☐ Payment Enclosed ☐ Bill me

☐ Check ☐ Money Order ☐ MasterCard ☐ VISA

Name of Cardholder _____

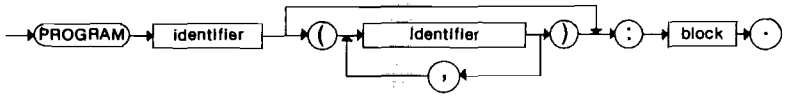
MC# and Interbank#/VISA# _____

Exp. Date _____

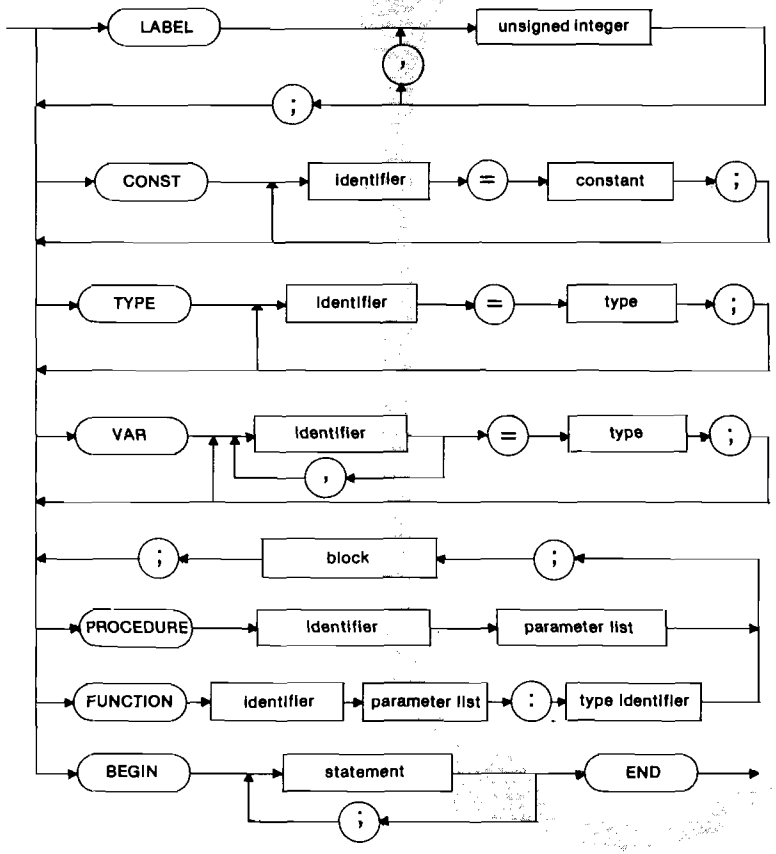
Signature _____

These tables summarize the rules (syntax) for writing Pascal programs. These rules define most of the UCSD Pascal language In standard use. The ovals containing upper case are Pascal keywords, the lower case entries are programmer supplied names.

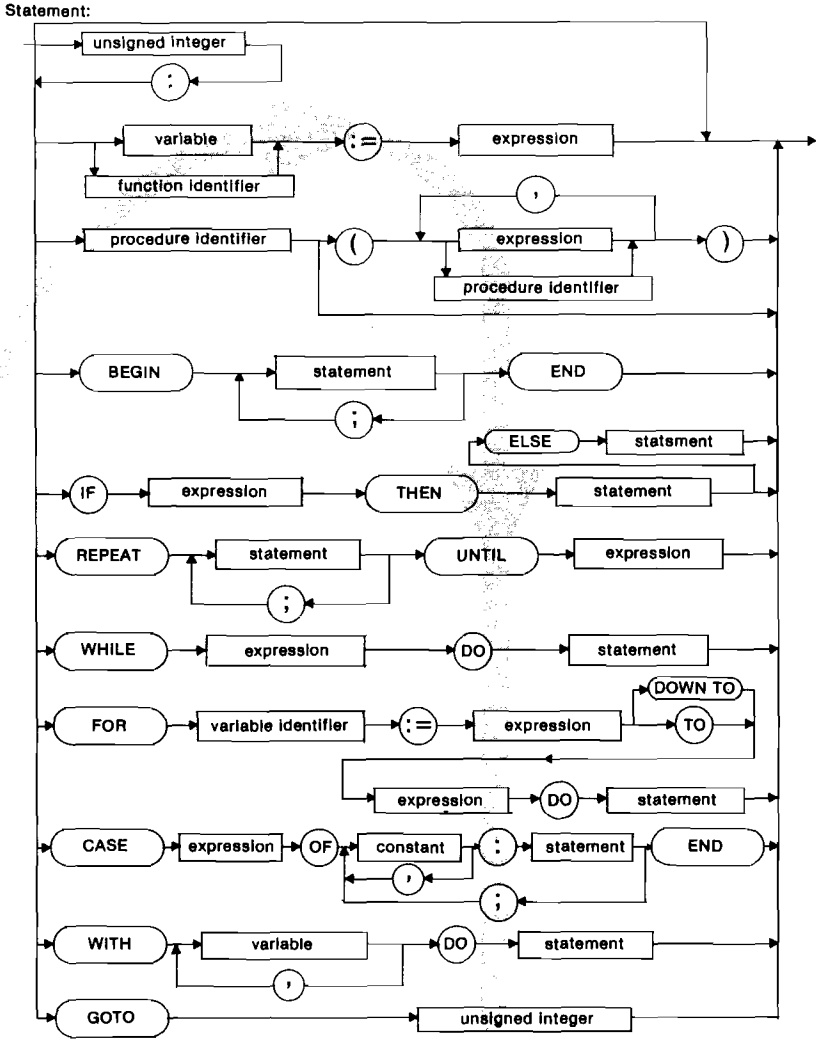
Program:



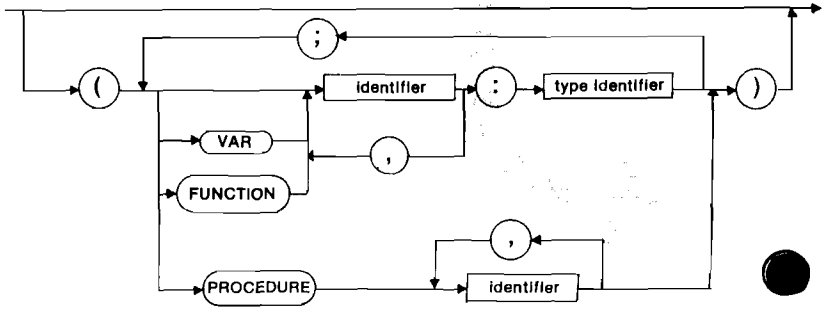
Block:



Statement:



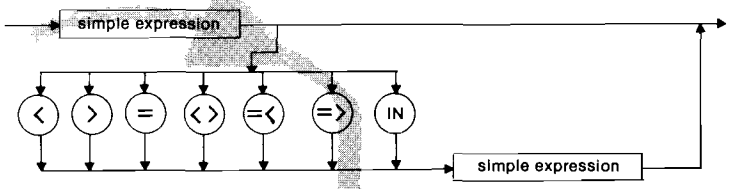
Parameter List:



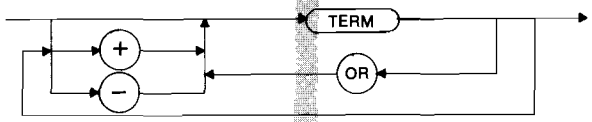
NOTE: Words or symbols that appear in ovals are required, objects that appear in rectangles are defined in other diagrams.

PASCAL
MICRO™ Data Sheet #13

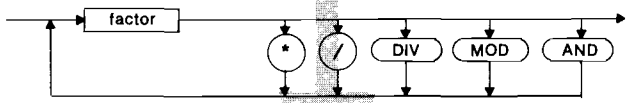
Expression:



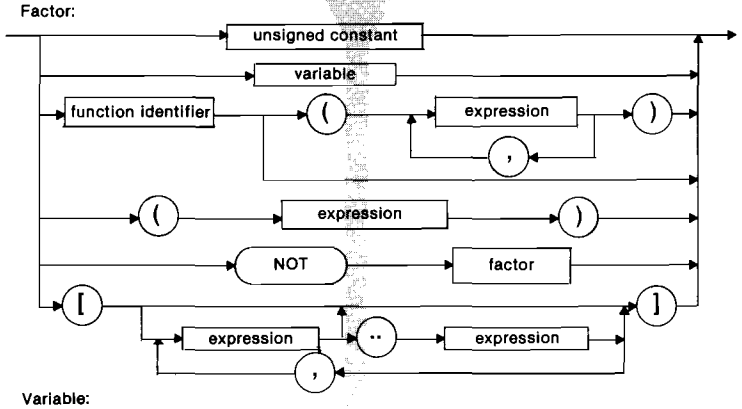
Simple Expression:



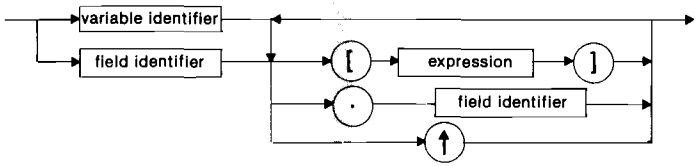
Term:



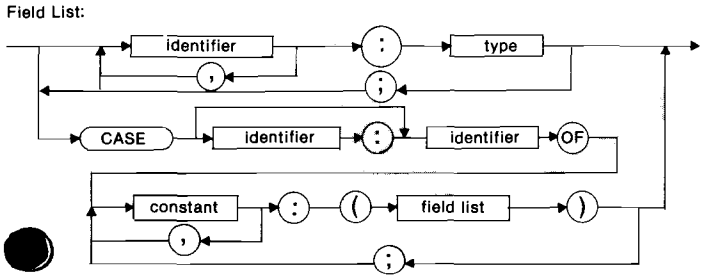
Factor:



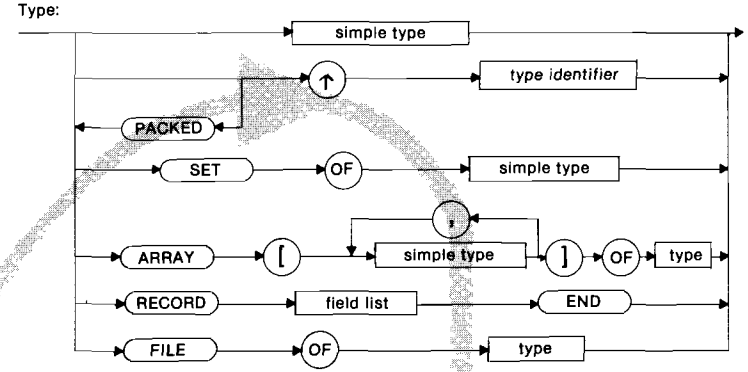
Variable:



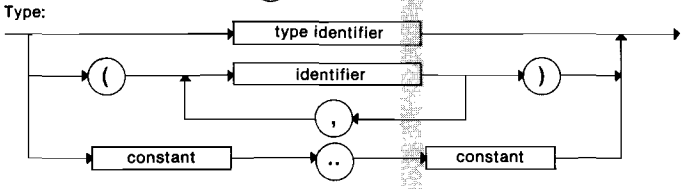
Field List:



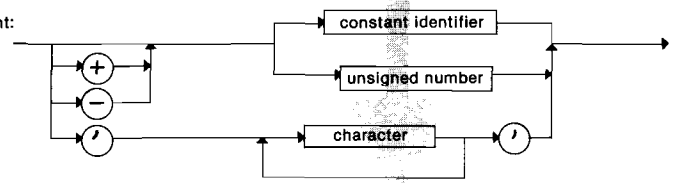
Type:



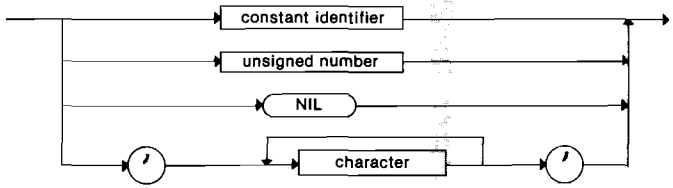
Simple Type:



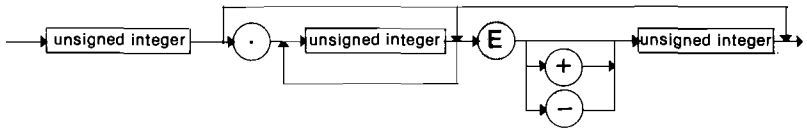
Constant:



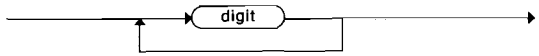
Unsigned Constant:



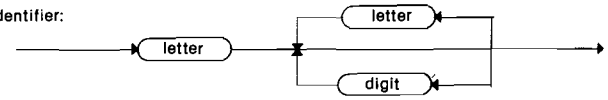
Unsigned Number:



Unsigned Integer:



Identifier:



MICRO™ Data Sheet #13
PASCAL

NATIONAL ADVERTISING REPRESENTATIVES

WEST COAST

The R.W. Walker Co., Inc.
Gordon Carnie
2716 Ocean Park Boulevard
Suite 1010
Santa Monica, California 90405
(213) 450-9001

serving: Washington, Oregon, Idaho, Montana, Wyoming, Colorado, New Mexico, Arizona, Utah, Nevada, California, Alaska, and Hawaii (also British Columbia and Alberta, Canada).

MID-WEST TERRITORY

Thomas Knorr & Associates
Thomas H. Knorr, Jr.
333 N. Michigan Avenue
Suite 707
Chicago, Illinois 60601
(312) 726-2633

serving: Ohio, Oklahoma, Arkansas, Texas, North Dakota, South Dakota, Nebraska, Kansas, Missouri, Indiana, Illinois, Iowa, Michigan, Wisconsin, and Minnesota.

MIDDLE ATLANTIC AND SOUTHEASTERN STATES

Dick Busch Inc.
Richard V. Busch
6 Douglass Dr., R.D. #4
Princeton, NJ 08540
(201) 329-2424

Dick Busch, Inc.
Eleanor M. Angone
74 Brookline
E. Atlantic Beach, NY 11561
(516) 432-1955

serving: New York, Pennsylvania, New Jersey, Delaware, Maryland, West Virginia, Virginia, D.C., North Carolina, South Carolina, Louisianna, Tennessee, Mississippi, Alabama, Georgia, and Florida.

NEW ENGLAND

Kevin B. Rushalko
Peterboro, New Hampshire 03458
(603) 547-2970

serving: Maine, New Hampshire, Vermont, Massachusetts, Rhode Island, Connecticut, and Kentucky.

ADVERTISING MANAGER
Cathi Bland

address materials directly to:
MICRO INK, Advertising
34 Chelmsford Street
Chelmsford, Massachusetts 01824
(617) 256-5515

Advertiser's Index

Aardvark Technical Services, Ltd. . . . 39
ABC Data Products . . . 24
Acorn Software Systems . . . 84
Alternative Energy Products . . . 84
Anthro-Digital Software . . . 18
Armidillo Software . . . 102
A2 Devices . . . 63
Aurora Software . . . 27
Chameleon Computing . . . 53
Commander Micro Systems Specialties . . . 20
Compu $ense . . . 14,16,49,79,93
CompuTech . . . 70
Computer Case Co. . . . 91
Computer Mail Order . . . 56-57
Computer Science Engineering . . . 24
Datamost, Inc. . . . IFC, 77
Digital Acoustics . . . 61
Execom, Inc. . . . 70
Gimix, Inc. . . . 1
Gloucester Computer Bus Co. . . . 79
Gnosis . . . 64
Gooth Software . . . 15
Hayden Software . . . 4
Human Systems Dynamic . . . 76
Huntington Computing . . . 94
Intec Peripherals Corp. . . . 25
Interesting Software . . . 9
John Bell Engineering . . . 21
L Com . . . 55
Leading Edge . . . BC
Logical Devices . . . 20
Lyco Computing . . . 52
Nibble . . . 107
Manx Software . . . 76
MICRObits (Classifieds) . . . 88,93
Microcomputing . . . 32
MICRO INK . . . 15, 45
Micro Motion . . . 59
Micro Signal . . . 93
Micro-Ware Distributing, Inc. . . . 68
Modular Mining Systems . . . 87
Modular Systems . . . 25
Performance Micro Products . . . 69
Perry Peripherals . . . 60
Privac . . . 42
RC Electronics . . . 35
RH Electronics . . . 6, 80
Scientific Software . . . 88
SGC . . . 46
SJB Distributors . . . 30
Skyles Electric Works . . . 8
Smartware . . . 97
Softronics . . . 2
Softside Publications . . . 108
Sorrento Valley Assoc. . . . 36
Talbot Microsystems . . . 59
Unique Data . . . 100
Versa Computing . . . 10
Vista Computing . . . IBC
XPS, Inc. . . . 99
Zytrex . . . 50

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

Next Month in MICRO

New Section for the Serious Novice!

Developing Computer Literacy — Appearing monthly in MICRO; basics for home and for school, in easy-to-understand terms. Learn how to develop computer literacy using the VIC-20, Commodore 64, Atari 400, TRS-80 CC, TI-99, Sinclair Timex, as well as the Apple, PET, or Atari 800. Look for

MICROCalc for VIC and all Commodore computers, Apple, and CC — a worksheet program to define and perform calculations.

DIGI-DRAFT — An Atari graphics program for drawing images on the screen to save on tape or disk for retrieval later.

BANNER: A Display Program for the CC — The scrolling screen displays any message of your choice.

The Computer Revolution — A look at the public's response to computers.

March: Printer Feature

MULTIC — Multic Column Print for the AIM
A Full Byte for Your Apple Printer
PRINT-USING on the Apple
Plotting with the VIC
Disk ID for OSI Printed Directories

Plus:

A Versatile Hi-Res Graphics Routine for the Apple
Animated Graphics on the 6809
Single Floppy Disk Interface for 6502

20% OFF

Your money goes farther when you subscribe. During the course of a year, when you subscribe, you save 20% (in the U.S.).

Pay only $24.00 ($2.00 a copy) for 12 monthly issues of MICRO sent directly to your home or office in the U.S.

More MICRO for Less Money When You Subscribe

But on the newsstand — if you can locate the issue you want — you pay $30.00 a year ($2.50 a copy).

Special Offer — Subscribe for 2 years ($42.00) and get 30% off the single issue price.

Subscribe to MICRO today.

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Please send me MICRO for __ 1 year __ 2 years
NOTE: Airmail subscriptions accepted for 1 year only.

Check enclosed $_____
Charge my _____ VISA account
_____ Mastercard account

No. _____

Expiration date _____

Name _____

Address _____

_____

City/State _____ Zip ____

Subscription Rates Effective January 1, 1982

Country	Rate
United States	$24.00 1 yr.
	42.00 2 yr.
Foreign surface mail	27.00
Europe (air)	42.00
Mexico, Central America, Mid East, N. & C. Africa	48.00
South Am., S. Afr., Far East, Australasia, New Zealand	72.00

* Airmail subscriptions accepted for only 1 year.
For U.S. and Canadian 2-year rates, multiply by 2.

Job Title: _____

Type of Business/Industry: _____

Vista DaVinci
GREAT INNOVATORS

15th century renaissance man, Leonardo Da Vinci, contributed an inexhaustible collection of inventions and ideas to solve the world's problems. These ideas were centuries ahead of their time. The studies on this page deal with experiments in manned flight.

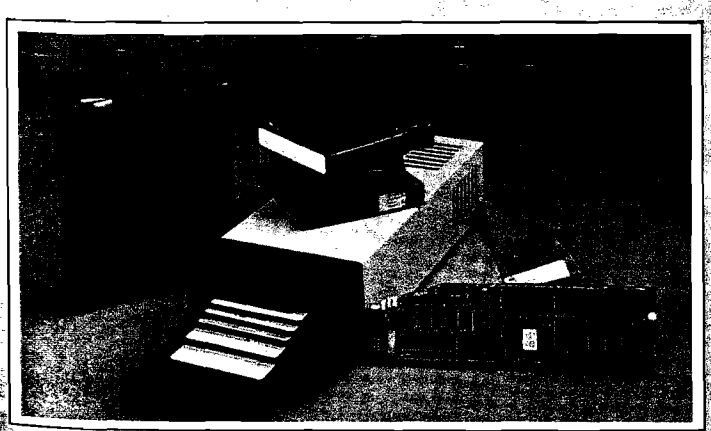
Like Leonardo, Vista Computer is answering many of today's complex computer storage problems. The Vista V1200 is a great solution to Apple II™ storage. Mass storage for your Apple II™ Computer has always been a problem. On one hand, there were the exotic, expensive hard disks with no cost efficient means of backup. On the other hand, the Apple floppy drive lacked the speed and storage demanded by today's professionals.

Vista's V1200 offers both at an incredibly attractive price. The removable VistaPak cartridges offer 6 Megebytes of removable storage each and can be backed up like a floppy.

Now hard disk storage and speed can be yours with the added capability of interchangeable media. The V1200 eliminates the worries of head crashes, drive alignments, lost data, or backup with a new application of field-proven floppy technology.

The VistaPak cartridges hold 6MB of formatted data each. The removable cartridge allows you to keep duplicates of your valuable data as well as to keep separate paks for your accounting, word processing, spread sheet and other applications. No other storage device offers more in flexibility and capability.

• Microprocessor controlled drive • DMA Controller • Removable Data Cartridges • CP/M, DOS & Pascal compatible • Quickcharge™ DOS enhancement included • Includes 1 VistaPak cartridge • Vista 120 Day Warranty



Vista V1200

™Apple II is a registered trademark of Apple Computer Co.

Contact Your Local Vista Dealer or Call our Vista Hotlines

Vista COMPUTER COMPANY INC.

1317 East Edinger / Santa Ana, CA 92705
(714) 953-0323 / (800) 854-8017

DISTRIBUTORS / REPRESENTATIVES

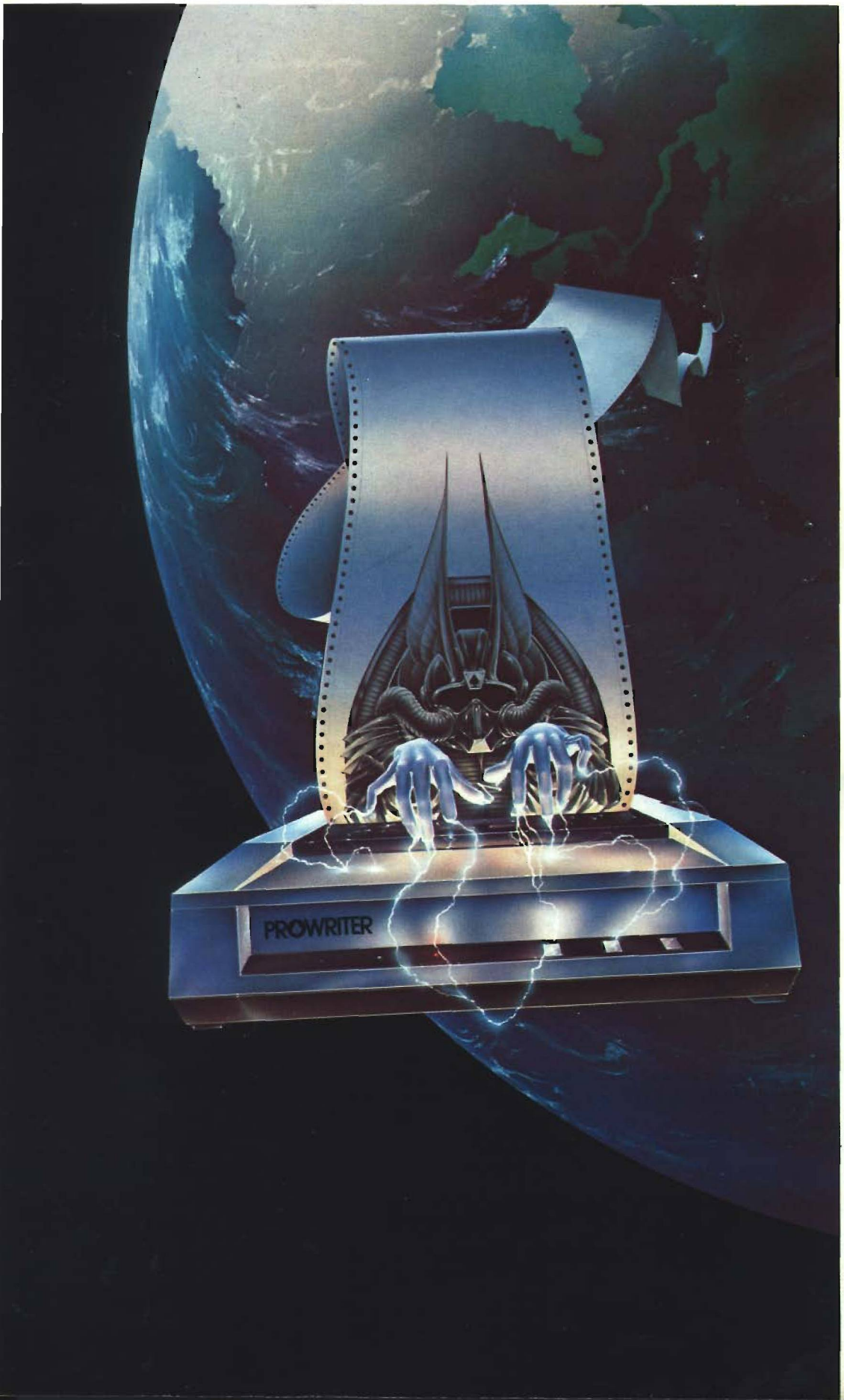
Western - Group 3 Wholesale
(213) 973-7844 (408) 727-1307

Northeast - Computers & Peripherals Inc.
(215) 426-6664

South Central - M.P. Systems
(214) 385-8865

THE PROWRITER COMETH.

(And It Cometh On Like Gangbusters.)



Evolution.

It's inevitable. An eternal verity.

Just when you think you've got it knocked, and you're resting on your laurels, somebody comes along and makes a dinosaur out of you.

Witness what happened to the Centronics printer when the Epson MX-80 came along in 1981.

And now, witness what's happening to the MX-80 as the ProWriter cometh to be the foremost printer of the decade.

SPEED

MX-80: 80 cps, for 46 full lines per minute throughput.
PROWRITER: 120 cps, for 63 full lines per minute throughput.

GRAPHICS

MX-80: Block graphics standard, fine for things like bar graphs.
PROWRITER: High-resolution graphics features, fine for bar graphs, smooth curves, thin lines, intricate details, etc.

PRINTING

MX-80: Dot matrix business quality.
PROWRITER: Dot matrix correspondence quality, with incremental printing capability standard.

FEED

MX-80: Tractor feed standard; optional friction-feed kit for about $75 extra.
PROWRITER: Both tractor and friction feed standard.

INTERFACE

MX-80: Parallel interface standard; optional serial interface for about $75 extra.
PROWRITER: Available standard—either parallel interface or parallel/serial interface.

WARRANTY

MX-80: 90 days, from Epson.
PROWRITER: One full year, from Leading Edge.

PRICE

Heh, heh.

Marketed Exclusively by Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021. Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.

LEADING EDGE.®

For a free poster of "Ace" (Prowriter's pilot) doing his thing, please write us.